(12) NACH DEM VERTRAG ÜBER DIE INTERNATIONALE ZUSAMMENARBEIT AUF DEM GEBIET DES PATENTWESENS (PCT) VERÖFFENTLICHTE INTERNATIONALE ANMELDUNG

(19) Weltorganisation für geistiges Eigentum
Internationales Büro

(43) Internationales Veröffentlichungsdatum
11. November 2004 (11.11.2004)

PCT

(10) Internationale Veröffentlichungsnummer
WO 2004/097734 A2

(51) Internationale Patentklassifikation[7]: G06N 7/00

(21) Internationales Aktenzeichen: PCT/EP2004/003561

(22) Internationales Anmeldedatum:
3. April 2004 (03.04.2004)

(25) Einreichungssprache: Deutsch

(26) Veröffentlichungssprache: Deutsch

(30) Angaben zur Priorität:
103 19 435.5 25. April 2003 (25.04.2003) DE

(71) Anmelder *(für alle Bestimmungsstaaten mit Ausnahme von US)*: **PEETERS, Bernd** [DE/DE]; Wilmersdorferstrasse 1, 21502 Geesthacht (DE). **RESAS, Uwe** [DE/DE]; Wilhelm-Hillmer-Strasse 1, 21339 Lüneburg (DE). **WEBER, Stefan** [DE/DE]; Bernhard-Riemann-Strasse 11, 21335 Lüneburg (DE).

(71) Anmelder und
(72) Erfinder: **HARDER, Wulf** [DE/DE]; Worther Weg 106, 21502 Geesthacht (DE).

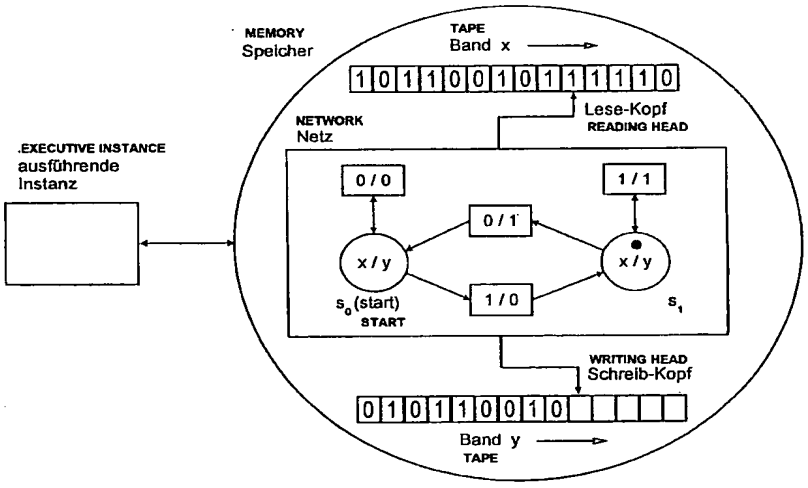
(74) Anwalt: **SIEMONS, Norbert**; Neuer Wall 41, 20354 Hamburg (DE).

(81) Bestimmungsstaaten *(soweit nicht anders angegeben, für jede verfügbare nationale Schutzrechtsart)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE,

[Fortsetzung auf der nächsten Seite]

(54) Title: DATA PROCESSING METHOD

(54) Bezeichnung: VERFAHREN ZUR VERARBEITUNG VON DATEN



(57) **Abstract:** The invention relates to a data processing method consisting in encoding, recording, reading and in processing a Petri net by at least one instance. The transitions of the Petri net extract at least one symbol tape or a symbol chain with the aid of at least one head and/or write them on at least one tape. In a variant, co-operating data processing networks are composed, the composition results are encoded and recorded in a memory, read therefrom and processed by at least one instance. Components can comprise cryptological functions. The data processing networks can receive second data of a cryptological function which is processed in a protected manner. Said invention makes it possible to carry out data processing without semantic analysis of written processing steps, is possible with less number thereof and enables a link of the processing steps to a piece of hardware which is difficult to disconnect.

(57) **Zusammenfassung:** Ein Verfahren zur Verarbeitung von Daten, bei dem ein Petri-Netz kodiert, in einen Speicher geschrieben und aus dem Speicher von wenigstens einer Instanz gelesen und ausgeführt wird, wobei Transitionen des Petri-Netzes Symbole oder Symbolketten mit Hilfe wenigstens eines Kopfes von wenigstens einem Band lesen und/oder auf wenigstens

[Fortsetzung auf der nächsten Seite]

WO 2004/097734 A2

KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) **Bestimmungsstaaten** *(soweit nicht anders angegeben, für jede verfügbare regionale Schutzrechtsart)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), eurasisches (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), europäisches (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Veröffentlicht:

— *ohne internationalen Recherchenbericht und erneut zu veröffentlichen nach Erhalt des Berichts*

Zur Erklärung der Zweibuchstaben-Codes und der anderen Abkürzungen wird auf die Erklärungen ("Guidance Notes on Codes and Abbreviations") am Anfang jeder regulären Ausgabe der PCT-Gazette verwiesen.

einem Bank schreiben (Fig. 1). In einer Alternative werden datenverarbeitende, kooperierende Netze komponiert, das Kompositionsergebnis kodiert, in einem Speicher geschrieben und aus dem Speicher von wenigstens einer Instanz gelesen und ausgeführt. Dabei können Komponenten kryptologische Aufgaben haben. Die datenverarbeitende Netze können von einer geschützt ausgeführten kryptologischen Funktion zweite Daten empfangen. Die Erfindung ermöglicht die Verarbeitung von Daten, die eine Semantik-Analyse offengelegter, möglicherweise weniger Verarbeitungsschritte verhindert und eine schwer trennbare Verknüpfung der Verarbeitungsschritte mit einer Hardware herstellen kann.

Verfahren zur Verarbeitung von Daten

Die Erfindung bezieht sich auf ein Verfahren zur Verarbeitung von Daten.

TECHNISCHER HINTERGRUND

Software und Inhalte, wie beispielsweise Musik, können heute über das Internet kostengünstig verbreitet werden. Es wurde eine Vielzahl von Kopierschutzmechanismen zur Durchsetzung von Urheberinteressen entwickelt. Der Schutz von Inhalten ist schwierig, wenn der Konsument der Mensch ist, der die Inhalte in unverschlüsselter Form erwartet. Während des Konsumierens kann jeder Inhalt in analoger Form aufgezeichnet, digitalisiert und dann kopiert werden. Dieses Problem wird *analogue hole* genannt. Das illegale Kopieren von Inhalten kann nach heutigem Stand der Technik nur behindert, aber nicht verhindert werden. Das Problem des analogue hole gibt es nicht bei der Ausführung von Software. Zwar ist der Benutzer einer Software der Mensch; der direkte "Konsument" ist aber ein Computer oder ein Prozessor. Deshalb ist es möglich, die Software ohne störende Auswirkungen für den Benutzer durch Kopierschutzmechanismen zu verändern. Unter dem Begriff *Softwareschutz* soll der Schutz des intellektuellen Eigentums, das mit der Software verbunden ist, verstanden werden. Hierzu gehört das Wissen über das Anwendungsgebiet der Software, über spezifische Probleme dieses Gebiets und deren Lösungen, die in der Software umgesetzt werden. Auch alle Techniken zur Erstellung der Software, die problem- oder lösungsspezifisch sein können, gehören zum schützenswerten intellektuellen Eigentum. Oft will ein Urheber sein Wissen schützen und sicherstellen, dass nur er die Software weiterentwickeln kann. Zum Schutz der genannten Werte ist es notwendig, eine Rekonstruktion (*Reverse-Engineering*) des Quellkodes oder eines äquivalenten Programmkodes mit Hilfe einer Analyse des Maschinenkodes der Software zu verhindern oder sicherzustellen, dass der Aufwand der Analyse größer ist als die Entwicklung der Software. Softwareschutz kann die Verhinderung einer unauthorisierten Benutzung der Software einschließen. Für diesen Zweck geschaffene Schutzverfahren und -vorrichtungen nennt man *Softwarekopierschutz*, obwohl viele Verfahren und Vorrichtungen nicht das Kopieren verhindern, sondern die Benutzung der Software. Dadurch wird ein illegales Kopieren demotiviert.

STAND DER TECHNIK

Bei bekannten Softwarekopierschutzverfahren werden Daten aus der Applikation an eine geschützte Hardware übertragen, von der Hardware verschlüsselt und anschliessend von der Applikation entschlüsselt oder mit vor der Compilierung der

Applikation verschlüsselten Daten verglichen. Nur bei korrekter Verschlüsselung der Daten durch die Hardware arbeitet die Applikation korrekt. Eine Methode zur Umgehung dieses Schutzes ist das Entfernen des Vergleiches der Daten aus dem Maschinenkode. Hierfür gibt es Gegenmassnahmen: Die Applikation überprüft sporadisch die Integrität des Maschinenkodes, so dass ein Angreifer auch diese Überprüfungen finden muss. Ein einfacherer Weg zur Umgehung des Softwarekopierschutzes ist daher folgender: Der Angreifer beobachtet die Kommunikation zwischen der Applikation und der Hardware. Er erstellt eine Tabelle mit den ausgetauschten Daten und simuliert die Hardware mit Hilfe dieser Tabelle in einem neu erstellten Hardware-Treiber. Eine Gegenmaßnahme ist die Implementierung vieler Kommunikationsvorgänge mit der Hardware. Dabei werden auch Zufallszahlen an die Hardware gesendet, so dass eine Tabelle zu groß werden würde. Die von der Hardware verschlüsselten Werte werden von der Applikation nur zum Schein verarbeitet. Entweder ist der Angreifer jetzt überzeugt, dass er die oben genannten Integritäts-Überprüfungen finden und entfernen muss, oder er beobachtet die Kommunikationsvorgänge. Er schreibt jeden an die Hardware gesendeten Wert in einen großen Speicher und prüft, welche Werte mehrfach gesendet werden. Nur diese Werte werden mit hoher Wahrscheinlichkeit von der Applikation wirklich verarbeitet. Die Hardware-Simulation benutzt dann eine Tabelle mit diesen Werten. Wird in der Simulation zu einer bestimmten Eingabe kein Eintrag in der Tabelle gefunden, wird mit einer Zufallszahl geantwortet, denn die Applikation könnte die Streuung der Antworten auswerten und erkennen. Die Tabelle ist deutlich kleiner als der zuvor benutzte Speicher.

In dem in [4] beschriebenen Softwarekopierschutzverfahren wird in einem mit dem PC verbundenen Steckerbauteil und parallel dazu in der geschützten Applikation eine Kryptofunktion berechnet. Teilfunktionen dieser Kryptofunktion können in die Applikation an verschiedenen Stellen eingefügt werden, so dass eine Extraktion ohne eine Semantik-Analyse des Programmkodes nicht möglich ist. Mit Hilfe der Ausgabewerte der Kryptofunktion werden Berechnungen der Software verfälscht und kurz vor einer schädlichen Auswirkung auf den Ablauf der Applikation mit Hilfe der Ausgabewerte des Steckerbauteils korrigiert. Ohne das für einen Angreifer nicht reproduzierbare Steckerbauteil ist die geschützte Applikation nicht benutzbar. Das beschriebene Verfahren hat den Nachteil, dass die Integration der Teilfunktionen in die zu schützende Software sehr aufwendig ist.

Bei einem weiteren Softwarekopierschutzverfahren werden Teile des zu schüt-

zenden Programms in einem nicht auslesbaren Speicher einer Smartcard gehalten und von dem Smartcardcontroller ausgeführt. Die Übertragung dieser Teile erfolgt nur in verschlüsselter Form. Beispiele für solche Prozessoren sind Vorrichtungen in Form von USB-Geräten der Firmen Syncrosoft [12] und Sospita [11]. Die Verschlüsselung der Software bei diesem Verfahren verhindert auch ein Reverse-Engineering.

Ein Nachteil der Ausführung des Programmkodes in einem speziellen, möglicherweise extern an den PC angeschlossenen Prozessor liegt in dem schlechten Durchsatz. Zwar ist diese für die Ausführung von Digital-Right-Management-Transaktionen ausreichend, jedoch ist die Ausführung von wesentlichen Teilen einer zu schützenden Applikation in vielen Fällen zu langsam. Die Integration von Vorrichtungen zur Programmentschlüsselung und geschützten Ausführung in einen PC-Prozessor wäre sehr teuer und würde zu Standardisierungs- und Kompatibilitäts-Problemen bei der Entwicklung und Verbreitung neuer Prozessor-Versionen führen.

In [10] wird ein Verfahren gegen Reverse-Engineering von Software beschrieben, das logische Verbindungen zwischen elementaren Operationen des Programms und Datenflüssen durch Einführung komplexer Adressierungsmechanismen verschleiert. Ein Nachteil dieser Erfindung zeigt sich bei dem Versuch, objekt-orientiert entwickelte Software zu schützen. Objekt-orientiert entwickelte Software enthält im allgemeinen sehr kurze Methoden, die aus wenigen Programminstruktionen bestehen und meist sehr einfache Datenflüsse mit einer kleinen Anzahl von Variablen realisieren. Wenigstens in diesem Fall ist das beschriebene Verfahren nicht wirkungsvoll. Weiter ist für dieses Verfahren keine Möglichkeit bekannt, die eine schwer trennbare Verknüpfung mit einer Hardware herstellt und somit das Kopieren der transformierten Software verhindert.

AUFGABE

Der vorliegenden Erfindung liegt die Aufgabe zugrunde, Verfahren zur Verarbeitung von Daten zu schaffen, die eine Semantik-Analyse offengelegter, möglicherweise weniger Verarbeitungsschritte erschwert oder verhindert und eine für einen Angreifer schwer trennbare Verknüpfung des Verarbeitungsverfahrens mit einer Hardware ermöglicht. Die Anwendung des Verfahrens auf vorgegebene Verarbeitungsschritte soll mit geringem Aufwand möglich sein.

LÖSUNG

Die Aufgabe wird durch die Ansprüche 1 und/oder 9 und/oder 20 gelöst.

Gemäss Anspruch 1 wird ein Petri-Netz kodiert, dessen Transitionen Symbole oder Symbolketten mit Hilfe eines oder mehrerer Köpfe mit wenigstens einem Band austauschen. Die Kodierung des Petri-Netzes wird in einen Speicher geschrieben und von wenigstens einer Instanz gelesen und ausgeführt. Petri-Netze und die Begriffe "Stelle", "Transition" und "Markierung" werden in [6] und [8] beschrieben. Die Begriffe "Kopf" und "Band" werden in Anlehnung an die Begriffe, die eine Turing-Maschine beschreiben, benutzt, wobei das Band aus technischen Gründen im Unterschied zum Modell der Turing-Maschine endlich ist. Turing-Maschinen werden beispielsweise in [5] beschrieben. Vorzugsweise wird bei jedem Lese- und Schreibvorgang der Kopf auf dem Band bewegt. Die Kopfbewegung kann aber auch steuerbar sein. Weiter ist das Vorhandensein von wenigstens zwei Köpfen für die Arbeitsgeschwindigkeit vorteilhaft, weil die meisten Operationen mit wenigstens zwei Operanden arbeiten. Ein Band kann ein Register eines Prozessors oder eine Speicherzelle eines RAMs sein. Ein Kopf kann ein Register mit einer Maske zur Maskierung von Werten des Bandes sein. Unter der Ausführung eines Petri-Netzes soll hier das Schalten von Transitionen des Petri-Netzes verstanden werden. Durch die Ausführung des Petri-Netzes, das auf Bändern arbeitet, werden Daten verarbeitet. Der Speicher und die ausführende Instanz bzw. die ausführenden Instanzen kann bzw. können auf viele Weisen gestaltet werden. Für den Erfindungsgedanken ist wichtig, dass die Semantik, die hinter dem Petri-Netz steht, auch bei Kenntnis des Petri-Netzes schwer analysierbar ist. Die Erzeugung und Kodierung des Petri-Netzes erfolgt vorzugsweise in einem anderen Speicher als die Ausführung. Die Kodierung des Petri-Netzes, der Köpfe, Bänder, Felder und Symbole ist in vielen Varianten möglich. Ein Angreifer, der Kenntnis über die Semantik des Petri-Netzes erlangen möchte, hat nur die Möglichkeit, das Petri-Netz mit den ihm bekannten Petri-Netzen zu vergleichen, oder die Semantik mit Hilfe von Ein- und Ausgabe-Beispielen zu erraten. Nach Anspruch 20 kann das Petri-Netz Symbole oder Symbolketten von einer kryptologischen Funktion empfangen und verarbeiten. Die kryptologische Funktion kann mit der das Petri-Netz ausführenden Vorrichtung fest verbunden sein, so dass eine für einen Angreifer schwer trennbare Verknüpfung des Verarbeitungsverfahrens mit einer Hardware hergestellt ist.

In einer Ausgestaltung der Erfindung bildet das Petri-Netz, der Kopf oder die Köpfe und das Band oder die Bänder eine universelle Turing-Maschine. Ein Petri-Netz kann die endliche Kontrolle der Turing-Maschine bilden. Auf dem Band der universellen Turing-Maschine ist die Kodierung einer Turing-Maschine

oder einer universellen Turing-Maschine gespeichert. Im letzten Fall kann wiederum eine Turing-Maschine oder eine universelle Turing-Maschine auf dem Band der zuletzt genannten universellen Turing-Maschine gespeichert sein usw. Diese Rekursion kann weitergeführt werden. Eine Semantik-Analyse der Verarbeitungsschritte bei der Ausführung des Petri-Netzes wird mit jeder Rekursion zunehmend erschwert.

In einer weiteren Ausgestaltung des Verfahrens tauscht das Petri-Netz mit einem bzw. mehreren weiteren Petri-Netzen über Kanäle Symbole oder Symbolketten aus. Hiermit lässt sich die Komplexität erhöhen und damit die Analysierbarkeit erschweren.

Das Schalten von Transitionen kann gemäss einer weiteren Ausgestaltung der Erfindung mit Hilfe von Tabellen schnell ausgeführt werden. In Analogie zu beispielsweise in [2] beschriebenen sequentiellen Maschinen kann aufgrund einer Markierung bzw. eines Zustands und einer Eingabe eine Folgemarkierung bzw. ein Folgezustand und eine Ausgabe aus einer Tabelle schnell ermittelt werden. Die Eingaben bzw. Ausgaben können auch optional erfolgen.

Eine Geschwindigkeitssteigerung beim Schalten der Transitionen lässt sich durch ein Verfahren erzielen, bei der ein Prozessor das Schalten einer Transition mit einer Instruktion ausführt, wobei eine Instruktion die Tabellen als Operand einliest. Der Instruktionssatz eines Prozessors kann mehrere solche Instruktionen enthalten.

Die Ausgaben eines Petri-Netzes können in ein weiteres Petri-Netz eingegeben und weiter verarbeitet werden. Ein aus mehreren Petri-Netzen bestehende System ist eine *Kooperation*. In einer weiteren Ausgestaltung der Erfindung bildet eine Kooperation von Petri-Netzen eine Turing-Maschine. Die Felder, Bänder und die endliche Kontrolle der Turing-Maschine werden als Petri-Netze kodiert, die über Kanäle Symbole oder Symbolketten austauschen und sich synchronisieren können.

Vorteilhaft für den Schutz von Software ist die Übersetzung dieser Software in ein Petri-Netz oder eine Kooperation von Petri-Netzen bzw. in eine Turing-Maschine. Dieser Übersetzungsvorgang könnte durch einen speziellen Compiler maschinell ausgeführt werden.

Die Ausführung einer Kooperation von Petri-Netzen kann in einer Ausgestaltung des Verfahrens durch die Ausführung einer Kompositionsvorschrift erfolgen. Dabei wird ein Petri-Netz erzeugt, das das gleiche äussere Ein-/Ausgabeverhalten

zeigt, wie die Kooperation der Petri-Netze, mit der Einschränkung, dass Ausgaben verzögert erfolgen können. Hierdurch wird die gewünschte Funktionalität des erzeugten Petri-Netzes nicht unbedingt beeinträchtigt.

Eine alternative Lösung der der Erfindung zugrunde liegenden Aufgabe sieht gemäss Anspruch 9 vor, dass datenverarbeitende, kooperierende Netze komponiert werden, das Kompositionsergebnis kodiert, in einen Speicher geschrieben und aus dem Speicher von wenigstens einer Instanz gelesen und ausgeführt wird, wobei das Kompositionsergebnis ein zu seinen Komponenten bezüglich des äusseren Ein-/Ausgabeverhaltens, ausgenommen Ausgabe-Verzögerungen, äquivalentes Netz ist. Hiervon ausgenommen wird ein Public Key Verschlüsselungsverfahren von [1] und [3], bei dem die Kompositionsergebnis einer Komposition endlicher Automaten einen Public Key bilden. Bei der vorliegenden Erfindung geht es um die allgemeine Verarbeitung von Daten unter Berücksichtigung der der Erfindung zugrunde liegenden Aufgabe. Die Aufgabe wird gelöst, weil eine Semantik-Analyse eines Kompositionsergebnisses ohne Kenntnis der Komponenten schwierig ist. Eine Dekomposition ist in vielen Fällen ein hartes resp. np-hartes Problem.

Das Kennzeichen des Anpruchs 9 schränkt nicht ein, welche Art von datenverarbeitenden, kooperierenden Netzen komponiert werden. Es ist bekannt, dass viele Netze einer Art sich durch Netze einer anderen Art simulieren lassen bzw. zueinander äquivalent sind. Beispielweise wurde in [7] und in [9] gezeigt, dass rekursive McCulloch-Pitts-Netze, eine spezielle Form künstlicher neuronaler Netze, zu endlichen Automaten äquivalent sind. Endliche Automaten lassen sich wiederum durch B/E-Netze beschreiben. B/E-Netze sind spezielle Petri-Netze. Eine Beschreibung der Komposition hängt naturgemäss von der formalen Definition der Netze ab, und es lassen sich unabhängig von dieser Definition inhaltlich viele verschiedene Varianten der Komposition definieren. Anspruch 9 beinhaltet auch Varianten von Kompositionen, die dem gleichen Erfindungsgedanken zugrunde liegen.

Die Komponenten und das Kompositionsergebnis können Petri-Netze sein, die Symbole oder Symbolketten über wahlweise vorhandene Kanäle senden und empfangen. In einer Ausgestaltung der Erfindung bildet jede Komponente eine sequentielle Maschine mit wahlweise mehreren Eingabekanälen und wahlweise mehreren Ausgabekanälen. Seien C eine aufzählbare Menge von Kanälen, Δ eine endliche Menge von endlichen Alphabeten, $\gamma : C \rightarrow \Delta$, $\Omega = (C, \Delta, \gamma)$ eine Kommunikationsregel,

$E_\Omega = \{e | e = \{(c,\sigma) | \sigma \in \gamma(c) \wedge ((c,\sigma_1) \in e \wedge (c,\sigma_2) \in e \Rightarrow \sigma_1 = \sigma_2)\}\} \cup \{\emptyset\}$ eine Menge von Ein-/Ausgabe-Ereignissen und S eine endliche Menge von Zuständen. Ein System von sequentiellen Maschinen wird definiert als

$$M_\Omega := \{ (S, E_\Omega, \delta, \beta, s_0) | \delta : R \to S \wedge \beta : R \to E_\Omega \wedge R \subset S \times E_\Omega$$
$$\wedge (\forall [(s,x), y] \in \beta \, \forall (c_x, \sigma_x) \in x \, \forall (c_y, \sigma_y) \in y : c_x \neq c_y) \wedge s_0 \in S \}.$$

In die Kompositionsfunktion geht als Parameter eine Menge von Synchronisationskanälen ein. Die Transitionen der zu komponierenden Maschinen schalten abhängig von einem imaginären globalen Takt und es gibt keine Nebenläufigkeit. Ein "Rendezvous" zwischen Sender und Empfänger von Symbolen soll möglich sein, was voraussetzt, dass die Komponenten aufeinander warten können. Realisiert wird dies durch das Schalten einer "leeren Transition" der wartenden Maschine. Die leere Transition liest nichts und schreibt nichts. Solche Transitionen gibt es bei nichtdeterministischen Automaten mit λ-Bewegungen [5]. Die λ-Bewegungen werden hier ε-Bewegungen genannt. Bei den zu komponierenden nichtdeterministischen sequentiellen Maschinen als B/E-Netze gibt es mehrere mögliche Schaltfolgen oder serielle Prozesse [6]. Jede mögliche Schaltfolge entspricht einer komponierten sequentiellen Maschine. Die Kompositionsfunktion ist eine Abbildung in einer Potenzmenge von sequentiellen Maschinen. Sei $\Omega = (C, \Delta, \gamma)$ eine Kommunikationsregel und B mit $B \subseteq C$ eine Menge von internen Synchronisationskanälen. Die Komposition $comp_B : M_\Omega^n \to 2^{M_\Omega}$ wird definiert als

$$comp_B := \Big\{ \Big((K_1, ..., K_n), \widetilde{K}\Big) \Big|$$
$$(K_1, ..., K_n) = ((S_1, E_\Omega, \delta_1, \beta_1, s_{0_1}), ..., (S_n, E_\Omega, \delta_n, \beta_n, s_{0_n}))$$
$$\wedge \exists T = \{ ((x_1, ..., x_n), (y_1, ..., y_n), (s'_1, ..., s'_n), \widetilde{x}, \widetilde{y}) |$$
$$([(s_{0_1}, x_1), s'_1], ..., [(s_{0_n}, x_n), s'_n]) \in \delta_1 \times ... \times \delta_n$$
$$\wedge ([(s_{0_1}, x_1), y_1], ..., [(s_{0_n}, x_n), y_n]) \in \beta_1 \times ... \times \beta_n$$
$$\wedge \exists H_x = \bigcup_{i \in \{1,...,n\}} x_i \; \exists H_y = \bigcup_{i \in \{1,...,n\}} \beta_i(x_i) :$$
$$H_x \in E_\Omega \wedge H_y \in E_\Omega$$
$$\wedge \forall (c,\sigma) : (c \in B \Leftrightarrow (c,\sigma) \in H_x \cap H_y)$$
$$\wedge \widetilde{x} = H_x \setminus H_y \wedge \widetilde{y} = H_y \setminus H_x \}$$
$$\exists \widetilde{M'_\Omega} = \{ \widetilde{K'} | \exists ((x_1, ..., x_n), (y_1, ..., y_n), (s'_1, ..., s'_n), \widetilde{x}, \widetilde{y}) \in T :$$

$$\widetilde{K}' = comp_B\left([(S_1, E_\Omega, \delta_1, \beta_1, s_1'), \ldots, (S_n, E_\Omega, \delta_n, \beta_n, s_n')]\right) \} :$$

$$\widetilde{K} = \left(\widetilde{S}, E_\Omega, \widetilde{\delta}, \widetilde{\beta}, \widetilde{s_0}\right)$$

$$\wedge\, \widetilde{S} = (s_{0_1}, \ldots, s_{0_n}) \cup \bigcup_{\left(\widetilde{S}', E_\Omega', \widetilde{\delta}', \widetilde{\beta}', \widetilde{s_0'}\right) \in \widetilde{M}_\Omega'} \widetilde{S}'$$

$$\wedge\, \widetilde{\delta} = \{\ [((s_{0_1}, \ldots, s_{0_n}), \widetilde{x}), (s_1', \ldots, s_n')] \mid$$

$$((x_1, \ldots, x_n), (y_1, \ldots, y_n), (s_1', \ldots, s_n'), \widetilde{x}, \widetilde{y}) \in T\ \}$$

$$\cup \bigcup_{\left(\widetilde{S}', E_\Omega', \widetilde{\delta}', \widetilde{\beta}', \widetilde{s_0'}\right) \in \widetilde{M}_\Omega'} \widetilde{\delta}'$$

$$\wedge\, \widetilde{\beta} = \{\ [((s_{0_1}, \ldots, s_{0_n}), \widetilde{x}), \widetilde{y}] \mid$$

$$((x_1, \ldots, x_n), (y_1, \ldots, y_n), (s_1', \ldots, s_n'), \widetilde{x}, \widetilde{y}) \in T\ \}$$

$$\cup \bigcup_{\left(\widetilde{S}', E_\Omega', \widetilde{\delta}', \widetilde{\beta}', \widetilde{s_0'}\right) \in \widetilde{M}_\Omega'} \widetilde{\beta}'$$

$$\wedge\, \widetilde{s_0} = (s_{0_1}, \ldots, s_{0_n})\ \}.$$

Bei der Komposition nach dieser Definition gibt es zwei Arten von Kanälen: Die Menge der Kanäle A, in der jeder Kanal nur von einer Maschine genutzt werden und eine Menge von Synchronisations-Kanälen $\overline{B}$, in der jeder Kanal von wenigstens zwei Maschinen genutzt wird. Die Menge der Synchronisationskanäle $\overline{B}$ ist bei der Komposition in interne und externe Synchronisationskanäle zu unterteilen. $B \subseteq \overline{B}$ ist die Menge der Synchronisationskanäle, die nicht mehr in der komponierten Maschine genutzt werden. In vielen Anwendungen ist $B = \overline{B}$. Ein wesentlicher Unterschied zwischen internen und externen Synchronisationskanälen ist, dass eine Transition mit einem internen Synchronisationskanal nur schalten kann, wenn mit einer synchronisierten Transition über diesen Kanal ein Symbol ausgetauscht wird. Bei einem externen Synchronisationskanal ist die Bedingung nicht so scharf: Die Ein- oder Ausgabe auf dem Kanal darf nicht inkompatibel mit einer synchronisierten Transition sein. Die synchronisierte Transition muss demnach auch nicht mit dem Kanal arbeiten. Über einen externen Synchronisationskanal können mit der Aussenwelt Symbole ausgetauscht werden. Sollen Transitionen mit einem externen Synchronisationskanal intern synchronisiert werden, müssen zusätzliche interne Synchronisationskanäle eingerichtet werden. Ein möglicher rekursiver Kompositionsalgorithmus arbeitet folgendermassen: Eine Kompositionsroutine wird mit den Startzuständen der zu komponierenden Maschinen aufgerufen. In dieser Routine wird

die geordnete Menge der Startzustände als komponierter Zustand in eine Liste von komponierten Zuständen eingetragen. Dann wird eine Menge von kompatiblen Transitionen (je Komponente eine Transition) gesucht, die jeweils den Startzustand als Eingabestelle haben. Transitionen sind kompatibel, wenn alle diesen Transitionen zugeordneten Ereignismengen paarweise kompatibel sind und jedes Symbol, das von einer Maschine auf einem internen Synchronisationskanal geschrieben bzw. gelesen wird von einer anderen Maschine gelesen bzw. geschrieben wird. Zwei Ereignismengen sind kompatibel, wenn alle Ereignisse paarweise kompatibel sind oder wenigstens eine Menge leer ist. Die leere Ereignismenge ist zu jeder anderen Ereignismenge kompatibel. Zwei Ereignisse eines internen bzw. externen Synchronisationskanals sind kompatibel, wenn sie entweder verschiedene Kanäle betreffen oder das gleiche Symbol von der einen Maschine gelesen und von der anderen Maschine geschrieben bzw. von beiden Maschinen gelesen oder geschrieben wird. Es ist z.B. ausgeschlossen, dass zwei Maschinen verschiedene Symbole auf einem Kanal gleichzeitig schreiben. Für jede gefundene Menge von kompatiblen Transitionen wird die geordnete Menge der Ausgangsstellen der Transitionen als komponierter Zustand in die Liste der komponierten Zustände eingetragen und als Startzustände in einen rekursiven Aufruf der Kompositionroutine eingegeben, wenn der komponierte Zustand noch nicht in der Liste enthalten war. Falls der komponierte Zustand bereits in der Liste war, wird die Kompositionsroutine beendet. Der Algorithmus endet, wenn keine neuen komponierbaren Zustände mehr gefunden werden.

Die Information über die Nebenläufigkeit geht bei der Komposition *comp* verloren. Um Nebenläufigkeit zu berücksichtigen, muss die Kompositionsregel modifiziert werden. Bei der Komposition werden dann nur Transitionen mit Synchronisationskanälen zu einer Transition zusammengefasst.

Kompositionsergebnisse haben oft äquivalente Zustände. Wenn die Produkte Kompositionsergebnisse in weiteren Kompositionen weiterverarbeitet werden, sind solche Redundanzen unerwünscht. Daher ist man bestrebt, äquivalente Maschinen mit einer minimalen Anzahl von Zuständen zu finden. Die Abbildung der Minimierung wird im folgenden durch eine Funktion $min : M_\Omega \rightarrow M_\Omega$ bezeichnet.

Nach einer Komposition entstehen oft unerwünschte Transitionen mit leeren Ereignismengen. Diese Transitionen können durch eine stellenberandete Vergröberung (Erklärung des Begriffs in [6]) ersetzt werden, wobei die Ränder die Eingangs- und Ausgangsstelle der Transition sind. Dies wird so oft wiederholt,

bis es keine leeren Transitionen in der Maschine mehr gibt. Diese Abbildung wird im folgenden durch eine Funktion $red : M_\Omega \rightarrow M_\Omega$ bezeichnet.

Der Kontrollfluss und die Struktur einer Turing-Maschine kann verschleiert werden, indem einige Bestandteile der Turing-Maschine komponiert werden. Beispielsweise könnte man Gruppen von beliebigen Feldern verschiedener Bänder komponieren. Felder von Bändern können auch mit dem Programm oder Leseköpfen komponiert werden. Weitere Kombinationen auch mit weiteren Komponenten, die nicht Bestandteil der Turing-Maschine sind, sind denkbar.

In einer weiteren Ausgestaltung des Verfahrens werden zu komponierende, datenverarbeitende Netze durch eine Übersetzung von Algorithmen gebildet. Hierdurch wird eine Dekomposition und Analyse von Algorithmen erschwert oder verhindert.

In einer weiteren Ausgestaltung des Verfahrens ist wenigsten ein der zu komponierende, datenverarbeitende Netze eine kryptologische Komponente. Wenn diese Komponente zufällig erzeugt und geheim gehalten wird, ist eine Dekomposition des Kompositionsergebnis wesentlich erschwert oder unmöglich, insbesondere, wenn mehrere Komponenten kryptologische Komponenten mit verschiedenen Aufgaben sind. Dieses Verfahren ist geeignet, um Folgen von Operationen zu verschlüsseln. Eine Operation liest die Operanden und schreibt ein Ergebnis. Ein Angreifer, der Kenntnis über die Operation erlangen möchte, hat die Möglichkeit, das Netz, das die Operation repräsentiert, mit ihm bekannten Netze zu vergleichen oder er versucht, mit Hilfe von Ein- und Ausgabe-Beispielen ein Modell der Operation zu bilden, so dass die Operation und das Modell ein äquivalentes Ein-/Ausgabe-Verhalten zeigen. Beides wird verhindert, wenn die Werte verschlüsselt sind und verschlüsselt verarbeitet werden. Dies ist möglich, indem die Netze zur Entschlüsselung der Operanden mit dem Netz der Operation und einem Netz zur Verschlüsselung des Ergebnisses komponiert werden. Es müssen Werte mit der Außenwelt unverschlüsselt ausgetauscht werden können. Eingaben werden dann nicht entschlüsselt. Analog werden Ausgaben nicht verschlüsselt. Werden Petri-Netze benutzt und wird für die Ent- und Verschlüsselung ein Stromverschlüsselungsverfahren benutzt, das die Eingabe-Symbolketten nur in einer Richtung verarbeitet und mit jeder Transition ein Symbol einliest und ein Symbol schreibt, benötigt die Verschlüsselung keine zusätzliche Zeit, weil jede Transition des Kompositionsergebnisses eine Zusammenfassung von Transitionen der Komponenten ist. Die obere Schranke für die Anzahl der Zustände des Kompositionsergebnisses ist das Produkt aus der Anzahl der Zustän-

de jeder Komponente. Jede Ausgabe einer verschlüsselten Operation sollte eine individuelle Verschlüsselung besitzen, damit ein Angreifer nicht durch Probieren verschiedener Konkatenationen von Operationen auf die Funktionalität der Operationen schließen kann.

In weiteren Ausgestaltung des Verfahrens können Komponenten Daten dekomprimieren und/oder Wasserzeichen in die Daten einfügen. Ein Wasserzeichen ist ein Identifizierungsmerkmal oder ein Zertifikat, das Daten hinzugefügt wird, ohne dass die Nutzung dieser Daten behindert wird. Dieses Verfahren ist bei der Distribution von Daten, wie beispielsweise Audio- und/oder Videodaten, an viele Endverbraucher geeignet. Die Wasserzeichen können beim Endverbraucher bei der Entschlüsselung der Daten in die Daten eingefügt werden. Vorzugsweise ist die Entschlüsselung und das Wasserzeichen endverbraucher-individuell. Dabei muss die zuvor gemachte Verschlüsselung nicht zwangsläufig endverbraucher-individuell sein. Die Entschlüsselung kann an eine durch eine spezielle Hardware geschützte kryptologische Funktion, deren Funktionswerte endverbraucher-individuell sind, gekoppelt werden.

In einer weiteren Ausgestaltung des Verfahrens können Register in einer Registerbank zusammengefasst werden und dadurch miteinander verschränkt werden. Mit Verschränkung ist gemeint, dass ein Angreifer nicht einen Registerwert ändern kann, ohne den Wert eines anderen Registers der Bank zu ändern. Es lässt sich die Integrität der Registerinhalte für einen Zeitraum sicherstellen, in dem wenigstens ein für den korrekten Ablauf des Programms wesentlicher Wert in einem Register der Bank gespeichert ist. Ein wichtiger Baustein einer Schreiboperation ist eine Maschine, die im folgenden Kombinierer genannt wird. Ein Kombinierer bildet mehrere Datenströme verschiedener Kanäle, die jeweils einem Register der Registerbank zugeordnet sind, auf einen Datenstrom eines Kanals umkehrbar ab. Das Produkt der Beträge der Symbolmengen der eingehenden Datenströme ist eine obere Schranke für den Betrag der Symbolmenge der Ausgabe des Kombinierers. Es ist möglich, dass nicht alle Kombinationen von Symbolen auf den Eingabekanälen vorkommen. Der von einem Kombinierer erzeugte Datenstrom wird verschlüsselt. Dies ist günstiger, als die in den Kombinierer eingehenden Datenströme zu verschlüsseln, weil die Symbolmenge der Ausgabe des Kombinierers grösser als die Symbolmengen der Komponenten ist. Eine Stromverschlüsselung mit einer grösseren Symbolmenge ist effizienter als mit einer kleineren Symbolmenge. Zum Extrahieren der Daten eines Registers aus dem Datenstrom eines Kombinierers, wird der Datenstrom zunächst

entschlüsselt. Dann können nach Anwendung des Separators die Daten einzelner Register gelesen werden. Neben der Verschränkung von Registern hat das Speichern von Daten in einer Registerbank den Vorteil, dass Datenflüsse verborgen werden können. Es können nacheinander viele Operationen auf einer Registerbank arbeiten, ohne dass Zwischenresultate die Registerbank verlassen. Zur Verschleierung des Zustands einer Registerbank kann eine Pseudozufallszahl in ein Register geschrieben werden, die bei jedem lesenden und schreibenden Zugriff auf die Registerbank geändert wird. Der Pseudozufallszahlen-Generator ist dann eine Komponente der Registerbank. In diesen Generator können sich verändernde Daten aus Registern, die nicht in der Registerbank liegen, eingegeben werden.

In einer weiteren Ausgestaltung des Verfahrens empfängt eine kryptologische Komponente von einer geschützt ausgeführten Funktion Daten und verarbeitet diese, wobei das Kompositionsergebnis nicht oder fehlerhaft arbeitet, wenn von der kryptologischen Funktion keine oder fehlerhafte Daten empfangen werden. Hierdurch wird eine schwer oder nicht trennbare Kopplung des Kompositionsergebnisses mit der kryptologischen Funktion erreicht, was beispielsweise dafür geeignet ist, eine unauthorisierte Benutzung von Software zu verhindern, wenn das Kompositionsergebnis von der Software benötigt wird und das Ausführen der kryptologischen Funktion nicht frei reproduzierbar ist. Ein weiteres Verfahren sieht vor, dass ein weiteres in der Funktionalität eingeschränktes Kompositionergebnis die kryptologische Komponente nicht enthält und keine Daten von der kryptologischen Funktion empfangen braucht, um die fehlerfreie Funktionsweise zu gewährleisten. Dieses Verfahren eignet sich für die Distribution von Demonstrations-Versionen von Software, die frei kopiert und verteilt werden können. Das Kompositionergebnis muss in der Funktionalität eingeschränkt sein, damit ein Angreifer in der Vollversion der Software nicht das mit der kryptologischen Funktion gekoppelte Kompositionsergebnis durch das in der Demonstrations-Version benutzte Kompositionsergebnis ersetzt und so eine Vollversion ohne Einschränkungen herstellt.

Mit Hilfe eines alternativen Verfahrens wird erreicht, dass die Ausführung eines datenverarbeitendes Netz bzw. eines Programms an die ausführende Vorrichtung gekoppelt wird. Eine geschützt ausgeführte kryptologische Funktion, beispielsweise eine Funktion des TPM-Chips der Trusted Computing Platform Alliance (TCPA) [13], die mit der Vorrichtung, beispielsweise einem PC oder einem PDA, fest verbunden ist, tauscht Daten mit dem Netz bzw. dem Pro-

gramm aus. Das datenverarbeitende Netz bzw. das Programm arbeitet nicht oder fehlerhaft, wenn von der kryptologischen Funktion keine oder fehlerhafte Daten empfangen werden. In einer Ausgestaltung des Verfahrens wird ein Wert über die Berechnung eines Funktionswertes der kryptologischen Funktion hinaus für einen Angreifer nicht lesbar oder veränderbar gespeichert wird und bei einer weiteren Berechnung eines weiteren Funktionswertes beeinflusst dieser Wert das Ergebnis der weiteren Berechnung, wobei dieser Wert sich nach einer vorgegebenen Regel verändert. Hierdurch wird verhindert, dass mehrere Netz- bzw. Programm-Instanzen unkontrollierbar Funktionswerte der kryptologischen Funktion benutzen können.

BEISPIELE

In Fig. 1 hat eine ausführende Instanz Zugriff auf einen Speicher, der ein Petri-Netz und die Bänder x und y speichert. Die Anfangsmarkierung hat eine Marke auf einer Stelle, dem Startzustand s_0. Mit jedem Schalten einer Transition wird die Marke von der Eingangsstelle zur Ausgangsstelle bewegt, mit Hilfe eines Kopfes ein Symbol des Eingabealphabets vom Band x gelesen und mit Hilfe eines weiteren Kopfes ein Symbol des Ausgabealphabets auf Band y geschrieben. Nach jedem Lese- und Schreibvorgang bewegen sich die Köpfe ein Feld nach rechts. Das Netz führt eine binäre Multiplikation $[y = 2x]$ aus.

In allen folgenden Figuren wird auf die Darstellung der ausführenden Instanz, des Speichers, der Köpfe und der Bänder verzichtet. Anstatt des Begriffs "Petri-Netz" wird der Begriff "Netz" benutzt.

Fig. 2 zeigt eine binäre Addition. Eine Marke liegt auf dem Startzustand s_0. Die Transitionen tragen die Beschriftung der Form ab/c. Die Ein- und Ausgabekanäle werden in der Eingangsstelle jeder Transition in gleicher Form genannt. a und b sind die Kanäle für Operanden, c ist der Kanal für das Ergebnis. Transitionen, die die gleiche Eingangsstelle und die gleiche Ausgangsstelle haben, aber unterschiedliche Ein- oder Ausgaben, werden in dieser und vielen weiteren Darstellungen durch *ein* Rechteck dargestellt. Jede Zeile eines Rechtecks entspricht einer Transition.

Es gibt Fälle, in denen mehrere Ergebnisse einer Verarbeitung parallel berechnet und ausgegeben werden sollen. Fig. 3 zeigt ein Netz, das neben mehreren Eingabekanälen auch mehrere Ausgabekanäle besitzt und zwei binäre dargestellte, natürliche Zahlen addiert: $[c = a + b, d = a - b]$.

Weitere Beispiele von Netzen sind in Fig. 4 und 5 dargestellt. Das Netz in Fig.

4 berechnet [$t = 3 \cdot a, d = a - b$, $s = a + b$], das Netz in Fig. 5 [$s = a + b + c$].

Fig. 6 zeigt die Komposition von zwei Netzen M und M'. Die Ein- und Ausgabe-Ereignisse werden in den Transitionen durch Mengen wie in Anspruch 11 beschrieben. M schreibt mit Transition t_4 über den Kanal b das Symbol σ, das M' über den gleichen Kanal mit Transition t'_3 liest. b ist ein interner Synchronisationskanal. t_4 und t'_3 sind die einzigen Transitionen, die auf Kanal b arbeiten und können daher nur synchron schalten. Es gibt die Kompositionsergebnisse K_1, K_2 und K_3. Von den Stellen s_0 und s'_1 führen keine kompatiblen Transitionen weiter. Die beschriebene Kompositionsroutine endet daher nach dem Eintragen des Zustandes (s_0, s'_1) in die Liste der komponierten Zustände.

Fig. 7 zeigt die Komposition der gleichen Netze, bei der nur Transitionen mit Synchronisationskanälen zu einer Transition zusammengefasst werden, um die Information über die Nebenläufigkeit zu erhalten.

Will man in einer Maschine M einen Ausgabekanal a durch einen Kanal b bzw. einen Eingabekanal b durch einen Kanal a ersetzten, komponiert man M mit der in Fig. 8 dargestellten Maschine, wobei a bzw. b ein interner Synchronisationskanal ist. $\{\sigma_1, ..., \sigma_m\}$ ist die den Kanälen a und b zugeordnete Symbolmenge.

Fig. 9 zeigt, dass auch das Konkatenieren von Netzen durch eine Komposition ausgeführt werden kann. Transitionen mit leeren Ereignismengen ("leere Transitionen"), deren Eingangsstelle gleich der Ausgangsstelle ist, werden im folgenden *Wartetransitionen* genannt und in Fig. 9 durch leere Rechtecke dargestellt. M schreibt auf Kanal a eine 1 und dann auf Kanal b eine 2. M' schreibt auf Kanal c eine 3 und dann auf Kanal d eine 4. M bzw. M' haben außerdem einen Kanal k zum Konkatenieren und Wartetransitionen t_3 bzw. t'_1. In Transition t_2 von M wird auf Kanal k das Symbol κ geschrieben. Transition t'_2 von M' liest auf Kanal k das Symbol κ. Komponiert man M und M' mit k als internen Synchronisationskanal, so erhält man die Maschine K, die nacheinander auf Kanal a, b, c bzw. d eine 1, 2, 3 bzw. 4 schreibt. Wenn Maschinen konkatenierbar sein sollen, müssen die Umgebungen der Anfangs- und Endzustände entsprechend präpariert werden. Verschiedenen Transitionen können Kanäle zum Konkatenieren zugeordnet werden. Durch geeignete Substitutionen von Kanälen lassen sich die Konkatenationen beeinflussen.

In Fig. 10 wird die natürliche Zahl 2 durch eine Maschine repräsentiert, die die binäre Symbolkette 010 auf Kanal a ausgibt. Dies entspricht der Gleichung $a = 2$. Eine Maschine zur Addition $c = a + b$ liest auf Kanal a und b und schreibt

auf Kanal c. Beide Maschinen werden über den internen Synchronisationskanal a komponiert. Fig. 11 stellt das Ergebnis dar.

$$[c = b + 2 \; mod \; 8] = comp_{\{a\}} ([c = a + b], [a = 2])$$

Weil der Kompositionsalgorithmus im Endzustand von $a = 2$ keine Transition mehr findet, bricht er ab. Das Kompositionsergebnis kann nur Symbolketten der Länge 3 ausgeben, ausgedrückt durch $mod\, 8$. Komponiert man $c = b + 2 \; mod \; 8$ über den internen Synchronisationskanal b mit der Maschine $b = 3$ aus Fig. 12 so erhält man die Maschine $c = 5$, ebenfalls in Fig. 12 dargestellt.

$$[c = 5] = comp_{\{b\}} ([c = b + 2 \; mod \; 8], [b = 3])$$

Fig. 13 stellt den gesamten Vorgang dar. Das Ergebnis der Kompositionen $c = 5$ erhält man ohne das Zwischenergebnis $c = b + 2 \; mod \; 8$, indem man alle Komponenten in einem Schritt komponiert.

$$[c = 5] = comp_{\{a,b\}} ([c = a + b], [a = 2], [b = 3])$$

Die Maschine $c = a + b$ kann unendlich lange Symbolketten verarbeiten. Das Kompositionsergebnis nach Komposition mit einem Operanden kann dies auch, wenn der Operand durch eine zyklische Transition, die Nullen ausgibt, abgeschlossen wird. In Fig. 14 wird $a = 2$ binär mit führenden Nullen dargestellt. Das Kompositionsergebnis nach Komposition mit $c = a + b$ kann unendlich lange Symbolketten verarbeiten, ebenfalls in Fig. 14 dargestellt.

$$[c = b +^{\infty} 010] = comp_{\{a\}} ([c = a + b], [a =^{\infty} 010])$$

In einem weiteren Beispiel soll die Gleichung $a = 2a + c$ vereinfacht werden. Damit sie verarbeitet werden kann, wird sie als Netz kodiert. Dazu wird zunächst die Gleichung $d = 2a + c$ gebildet und anschliessend d durch a substituiert. Fig. 15 stellt den ersten Schritt dar. Wird d nun durch a substituiert, so sind alle Transitionen ungültig, in denen vor der Substitution die den Kanälen a und d zugeordneten Symbole ungleich waren. Die ungültigen Transitionen und eine nicht mehr erreichbare Stelle s_3 sind in Fig. 15 grau unterlegt. Nach der Substitution ist der Kanal a zu entfernen, weil in einer Transition nicht gleichzeitig auf einem Kanal gelesen und geschrieben werden kann. Damit nach dem Entfernen die Information des Kanals a nicht verloren ist, wurde die Kopie $a' = a$ angelegt. Fig. 16 stellt auf der Eingangsstelle der Transition min das Ergebnis dieser Schritte dar. Man kann zeigen, dass s_1 und s_2 äquivalent sind. Nach der Minimierung durch min ist das Ergebnis die Gleichung $a = -c$. Dies kann, wie

in Fig. 17 dargestellt, verifiziert werden. Wird bei einer binär arbeitenden sequentiellen Maschine $a = e + f$ der Übertragszustand als Startzustand definiert, erhält man die Maschine resp. das Netz $a = e + f + 1$. Dann wird e durch das Eins-Komplement von c ersetzt , so dass $a = -c + f$. Nach $f = 0$ ist $a = -c$.

Im folgenden werden Matrizen natürlicher Zahlen als Netz kodiert und durch Komposition addiert. Das Netz aus Fig. 18 entspricht einer Funktion von $\{1,2\}^2$ auf $\{0,1,2,3\}$, die die Zeile i und die Spalte j der Matrix

$$A = \begin{pmatrix} 2 & 0 \\ 1 & 3 \end{pmatrix}$$

liefert, wobei die Funktionswerte $\{0,1,2,3\}$ binär als $\{^{\infty}0,^{\infty}01,^{\infty}010,^{\infty}011\}$ kodiert sind. Das Netz liest zunächst den Zeilen- und Spaltenindex und gibt dann das entsprechende Matrix-Element aus. Die Bezeichnung der Kanäle zum Einlesen der Indizes spielt bei der Komposition mit anderen Netzen eine Rolle. Deshalb werden im folgenden in Abweichung zu üblichen Matrix-Schreibweisen die Indizes dem Matrix-Bezeichner hinzugefügt: A_{ij}. Die Komposition ergibt

$$\left[c_{ij} = \begin{pmatrix} 3 & 1 \\ 2 & 3 \end{pmatrix}\right] =$$

$$comp_{\{a,b\}}\left([c = a + b], \left[a_{ij} = \begin{pmatrix} 2 & 0 \\ 1 & 3 \end{pmatrix}\right], \left[b_{ij} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}\right]\right),$$

was in Fig. 19 dargestellt ist. Wartetransitionen werden durch die Zeichenfolge '//' in der Eingangsstelle (Ausgangsstelle) symbolisiert. Der Startzustand von $c = a + b$ hat eine Wartetransition. Die Kanäle i und j werden zu externen Synchronisationskanälen. Sie bleiben im Kompositionsergebnis erhalten, während die internen Synchronistionskanäle a und b nicht erhalten bleiben. c ist kein Synchronisationskanal.

Fig. 20 zeigt die Kanalstruktur einer Turing-Maschine, die durch eine Kooperation von Netzen gebildet wird. Die Transitionen entsprechen den beteiligten Kooperationspartnern, die Stellen entsprechen den Kanälen. Eine endliche Kontrolle, als Netz realisiert, liest über Kanal x bzw. schreibt über Kanal y Symbole auf ein Band. Die endliche Kontrolle gibt bei jedem Lese- und Schreibvorgang über den Kanal I Bewegungs-Anweisungen an den Kopf. Ein Kopf H mit Feldern F_i kommuniziert über die Kanäle z_i. Fig. 21 zeigt ein Band mit Feldern zum Speichern der Symbole 0, 1 und τ. Alle Felder speichern am Anfang das Symbol τ. In Fig. 22 ist eine Initialisierung $Init_{01}$ des Bandes mit der Symbolkette $\tau 01\tau$ dargestellt. Ein Punkt anstelle eines Symbols in einer Transition

bedeutet, dass jedes erlaubte Symbol hier eingesetzt werden darf. Das Transitionsrechteck mit der Beschriftung " $./R\tau$" und der Eingangsstelle mit der Beschriftung "x/Iy" ist die Kurzschreibweise für drei Transitionen mit gleicher Ein- und Ausgangsstelle mit den folgenden Eingabe-/Ausgabe-Ereignismengen: 1. Transition: $\{(x,0)\}$ / $\{(I,R),(y,\tau)\}$, 2. Transition: $\{(x,1)\}$ / $\{(I,R),(y,\tau)\}$, 3. Transition: $\{(x,\tau)\}$ / $\{(I,R),(y,\tau)\}$. Komponiert man die Initialisierung $Init_{01}$ mit dem Kopf H und vier Bandfeldern $F_1,...,F_4$, wobei die internen Synchronisationskanäle x , y, I, $z_1...z_4$ sind, so gibt es im ersten Kompositionsschritt von den Startzuständen aller Komponenten ausgehend nur *eine* Menge kompatibler Transitionen, wie in Tab. 1 dargestellt:

Komponente	Kanäle	Symbole
Initialisierung P	x/Iy	$\tau/R\tau$
Kopf H	yI/z_1	$\tau R/\tau$
Feld F_1	z_1/x	τ/τ
Felder F_2, F_3, F_4	(Wartetransitionen)	-

Tab. 1 Kompatible Transitionen

Die Komponenten bilden einen geschlossenen Kreislauf von Symbol-Produzenten und -Konsumenten. Nach der Komposition ist der Kopf auf Feld F_3, das eine Eins speichert, positioniert. Eine endliche Kontrolle P_1 zur Erkennung der Sprache $L = \{0^n1^n | n \geq 1\}$ ist in Fig. 23 dargestellt. Es wird vorausgesetzt, dass das zu erkennende Wort mit einem vorgehenden und nachfolgenden τ linksbündig auf dem Band steht und der Kopf auf ein Feld links neben dem rechten τ positioniert ist. Angenommen, das Wort gehört zur Sprache L, dann gibt es folgenden Ablauf (ähnlich wie in [5]): Die am weitesten rechts stehende Eins wird durch τ ersetzt. Dann bewegt sich der Kopf nach links bis zum nächsten τ, dann ein Feld nach rechts. Die am weitesten links stehende Null ist hier gespeichert und wird durch τ ersetzt. Dann wird die am weitesten rechts stehende Eins, dann die am weitesten links stehende Null durch τ ersetzt usw. Wenn eine Null durch τ ersetzt wurde, und rechts daneben ein τ gefunden wird, akzeptiert die Maschine das Wort. Das Akzeptieren wird der Außenwelt durch das Schreiben einer Eins auf dem Kanal Λ mitgeteilt. Findet die Maschine auf der Suche nach einer Null bzw. einer Eins eine Eins bzw. Null oder ein τ, wird das Wort nicht akzeptiert und eine Null auf Kanal Λ ausgegeben. Der Benutzer (in diesem Fall die endliche Kontrolle) des Bandes benötigt keine Kenntnis über die Struktur des Bandes. Das Band und die Felder könnten z.B. auch in einer Maschine komponiert sein.

Der Benutzer benötigt nur Kenntnis über die Schnittstelle des Bandes die Bedeutung der Ein- und Ausgabekanäle. Ein Benutzer der endlichen Kontrolle zur Erkennung der Sprache muß wissen, wie das zu prüfende Wort auf das Band zu schreiben ist. Dazu gehört das Wissen über die Bedeutung des Kanals I und die Konvention, ein vorausgehendes τ auf das erste Feld des Bandes zu schreiben. Dieses Wissen läßt sich durch die Maschine in Fig. 24 kapseln. Komponiert man diese Kapselung mit dem Band, kann man Symbolketten in der Form, wie in Fig. 25 dargestellt, eingeben. Sei $c \in L$. Konkateniert man E mit P_1 (Endzustand von E ist der Startzustand von P_1) und komponiert dies mit T, wobei alle von wenigstens zwei Maschinen benutzten Kanäle Synchronisationskanäle sind, so erhält man nach Anwendung von red eine Maschine, die in Fig. 26 a) dargestellt ist. Ist $c \notin L$, so ist das Ergebnis die in Fig. 26 b) dargestellte Maschine.

$$c \in L \wedge |c| = n \Leftrightarrow [\Lambda = 1] = red\left(comp_{\{c,x,y,I,z_1 \ldots z_{n+2}\}}\left(c, E, P_1, T\right)\right).$$

Fig. 27 a) zeigt die verschlüsselte Ausführung einer Operation. Die Operanden a, b bzw. c sind als a', b' bzw. c' verschlüsselt. Die Entschlüsselungen von a und b und die Verschlüsselung von c werden mit der Operation komponiert. Fig. 27 b) zeigt, wie unverschlüsselte Operanden a und b mit einer Operation verarbeitet werden und das Ergebnis c als c' verschlüsselt wird. Die Operation und die Verschlüsselung werden komponiert.

Fig. 28 a) bzw. b) stellt einen Kombinierer von Kanälen bzw. dessen Umkehrung, einen Separator, dar, der in einer Registerbank benutzt wird. In Fig. 29 wird ein möglicher Aufbau einer Registerbank dargestellt. Um einen Registerwert zu verändern, werden mehrere Operationen komponiert. Über den Kanal $\widetilde{x'}$ wird eine Registerbank R' mit drei Registern R_1, R_2 und R_3 gelesen. Der alte Registerwert von Register R_3 wird auf Kanal x_3 ausgegeben. Über den Kanal y_3 wird ein neuer Wert in das Register R_3 geschrieben. Fig. 29 b) zeigt eine Komposition, die den Inhalt von Register R_3 ausgibt, ohne die Registerbank zu verändern.

In Fig. 30 wird dargestellt, wie eine Hardware Daten der Applikation mit einer Stromverschlüsselung verschlüsselt. Der von der Hardware verschlüsselte Wert wird von der Applikation entschlüsselt. Die Entschlüsselungsfunktion wird mit einer Operation der Applikation komponiert. Das Ergebnis nach Ausführung der Operation ist verschlüsselt. Die Entschlüsselung mit Hilfe der Umkehrfunktion der Hardware und die Verschlüsselung mit der Operation geschehen parallel. Der

entschlüsselte Wert wird für einen Angreifer zu keinem Zeitpunkt sichtbar. Es spielt keine Rolle, ob die Hardware verschlüsselt und die Applikation entschlüsselt, oder umgekehrt. Wesentlich ist, dass die Verknüpfung der beiden Funktionen die Identität ist. Die parallele Entschlüsselung und Verschlüsselung ist möglich, weil nur Stromverschlüsselungen benutzt werden. Es können auch andere Kryptofunktionen benutzt werden. Viele bekannte Kryptofunktionen lassen sich nur mit Hilfe von Registern für das Speichern von Zwischenergebnissen, wie z.B. Rundenergebnissen, realisieren. Diese Zwischenergebnisse müssen in geschützten Registerbänken gespeichert werden. Wenn eine Runde einer Blockverschlüsselung eine Stromverschlüsselung ist, kann die letzte Runde dieser Blockentschlüsselung mit der Operation und den Umschlüsselungen der Operation komponiert werden. Anstelle einer Verschlüsselungsfunktion kann auch eine Hashfunktion in der Hardware ausgeführt werden. Fig. 31 zeigt ein mögliches Schema. Ein Teil der Eingabe in die Hashfunktion muß geheim bleiben. Dieser Teil entspricht dem Schlüssel bei einer Verschlüsselungsfunktion. Der andere Teil der Eingabe sind Daten aus der Applikation. Der Ausgabewert der Hashfunktion kann mit dem Ausgabewert einer Simulation der Hashfunktion z.B. durch sich kompensierende Operationen wie Addition und Subtraktion zu dem Ergebnis einer ausgeführten Operation addiert werden. Das Ergebnis einer solchen ausgeführten Operation ist nur dann korrekt, wenn die Hashfunktion und ihre Simulation den gleichen Wert liefern. Die Operation, die Hashfunktion resp. eine Runde der Hashfunktion, die Addition, Subtraktion und alle Umschlüsselungen der Operation sind zu komponieren. Der geheime Teil der Eingabe in die Hashfunktion und gegebenenfalls Zwischenergebnisse sind in geschützten Registerbänken zu speichern.

Sequentielle, reversible Maschinen können zur Verschlüsselung und Entschlüsselung benutzt werden. Ein Beispiel für eine sequentielle, reversible Maschine ist in Fig. 32 dargestellt. In jedem Zustand kann bei gegebener Ausgabe eindeutig die dazugehörige Eingabe bestimmt werden. Solche Maschinen mit einer deutlich höheren Anzahl von Zuständen, als in Fig. 32 beispielhaft dargestellt, eignen sich zur Komposition mit anderen Netzen, beispielsweise wie in Fig. 27 a) und b) dargestellt. Denkbar sind auch Maschinen mit verzögerten Ausgaben, wie in [3] beschrieben. Alle Maschinen können nicht-deterministisch erzeugt werden, z.B. mit Hilfe von Zufallszahlengeneratoren.

QUELLEN

[1] Feng Bao, Yoshihide Igarashi, Break Finite Automata Public Key Cryptosystem, *ICALP 1995: 147-158*, 1995.

[2] T.L. Booth, Sequential Machines and Automata Theory, John Wiley and Sons, 1967.

[3] Zongduo Dai, Dingfeng Ye, Kwok-Yan Lam, Weak Invertibility of Finite Automata and Cryptoanalysis on FAPKC, *Advances in Cryptology - ASIACRYPT'98: 227-241*, Springer, 1998.

[4] W. Harder, B. Peeters, Vorrichtung zum Schutz gegen unauthorisierte Benutzung von Software, Patent DE 3914233, 1990.

[5] J.E. Hopcroft, J.D. Ullman, Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie, 4. Auflage, Oldenbourg, 2000.

[6] E. Jessen, R. Valk, Rechensysteme, Grundlagen der Modellbildung, Springer, 1987.

[7] M. Minsky, Computation: Finite and Infinite Machines, Prentice-Hall, Englewood Cliffs, 1967.

[8] W. Reisig, Petri-Netze, Eine Einführung, Springer, 1982.

[9] R. Rojas, Theorie der neuronalen Netze, Springer, 1993.

[10] D.R. Wallace, System and method for cloaking software, US-Patent 6192475, 2001.

[11] Sospita, http://www.sospita.com, 2002.

[12] Syncrosoft Hard- und Software GmbH, http://www.syncrosoft.com, 2003.

[13] Trusted Computing Platform Alliance (TCPA), http://www.trustedcomputing.org, 2003.

ANSPRÜCHE

1. Verfahren zur Verarbeitung von Daten, dadurch gekennzeichnet, dass ein Petri-Netz kodiert, in einen Speicher geschrieben und aus dem Speicher von wenigstens einer Instanz gelesen und ausgeführt wird, wobei Transitionen des Petri-Netzes Symbole oder Symbolketten mit Hilfe wenigstens eines Kopfes von wenigstens einem Band lesen und/oder auf wenigstens einem Band schreiben.

2. Verfahren nach Anspruch 1, dadurch gekennzeichnet, dass das Petri-Netz, der Kopf oder die Köpfe und das Band oder die Bänder eine universelle Turing-Maschine bilden.

3. Verfahren nach Anspruch 1 oder 2, dadurch gekennzeichnet, dass wenigstens ein zweites Petri-Netz, insbesondere mit den Eigenschaften des in Anspruch 1 beschriebenen Petri-Netzes kodiert, in einen Speicher geschrieben und aus dem Speicher von wenigstens einer Instanz gelesen und ausgeführt wird, wobei Transitionen jedes Petri-Netzes Symbole oder Symbolketten über wenigstens einen wahlweise vorhandenen Kanal senden können, die von Transitionen anderer Petri-Netze über diesen Kanal oder diese Kanäle empfangen werden können.

4. Verfahren nach einem der Ansprüche 1 bis 3, dadurch gekennzeichnet, dass ein Petri-Netz auf eine Markierungs- bzw. Zustandsübergangstabelle und wahlweise wenigstens eine Ausgabetabelle oder eine Kombination aus beiden zugreift und hiermit in Abhängigkeit von der Markierung bzw. von dem Zustand und wahlweise abhängig von einer wahlweise vorhandenen Eingabe eine Folgemarkierung bzw. einen Folgezustand und wahlweise wenigstens eine Ausgabe ermittelt.

5. Verfahren zur Durchführung des Verfahrens nach Anspruch 4, dadurch gekennzeichnet, dass das Schalten der Transitionen eines Petri-Netzes von einem Prozessor ausgeführt wird, wobei der Prozessor wenigstens eine Prozessorinstruktion besitzt, die die Markierungs- bzw. Zustandsübergangstabelle und wahlweise wenigstens eine Ausgabetabelle oder eine Kombination aus beiden als Operanden verarbeitet.

6. Verfahren nach Anspruch 3, dadurch gekennzeichnet, dass eine Kooperation von Petri-Netzen eine Turing-Maschine bildet.

7. Verfahren nach einem der Ansprüche 3 und 6, dadurch gekennzeichnet, dass wenigstens ein Teil eines Programms in ein Petri-Netz oder eine Kooperation von Petri-Netzen übersetzt wird.

8. Verfahren nach einem der Ansprüche 3 bis 7, dadurch gekennzeichnet, dass die

Petri-Netze durch eine Kompositionsvorschrift ausgeführt werden, wobei ein zu den kooperierenden ersten und zweiten Petri-Netzen bezüglich des äusseren Ein-/Ausgabeverhaltens, ausgenommen Ausgabe-Verzögerungen, äquivalentes drittes Petri-Netz mit Hilfe des ersten und der zweiten Petri-Netze gebildet wird.

9. Verfahren zur Verarbeitung von Daten, ausgenommen auf der Komposition von endlichen Automaten basierende Public Key Verschlüsselungsverfahren, insbesondere in Verbindung mit einem der Ansprüche 1 bis 8, dadurch gekennzeichnet, dass datenverarbeitende, kooperierende Netze komponiert werden, das Kompositionsergebnis kodiert, in einen Speicher geschrieben und aus dem Speicher von wenigstens einer Instanz gelesen und ausgeführt wird, wobei das Kompositionsergebnis ein zu seinen Komponenten bezüglich des äusseren Ein-/Ausgabeverhaltens, ausgenommen Ausgabe-Verzögerungen, äquivalentes Netz ist.

10. Verfahren nach einem der Ansprüche 1 bis 8 und Anspruch 9, dadurch gekennzeichnet, dass die Komponenten und das Kompositionsergebnis Petri-Netze sind, wobei die Transitionen der Komponenten Symbole oder Symbolketten über wahlweise vorhandene Kanäle senden und empfangen können.

11. Verfahren nach Anspruch 8 oder 10, dadurch gekennzeichnet, dass die Petri-Netze sequentielle Maschinen M_Ω mit wahlweise mehreren Eingabekanälen und wahlweise mehreren Ausgabekanälen bilden, C eine endliche Menge von Kanälen, Δ eine endliche Menge von endlichen Alphabeten, $\gamma : C \rightarrow \Delta$, $\Omega = (C, \Delta, \gamma)$ eine Kommunikationsregel,

$$E_\Omega = \{e | e = \{(c,\sigma) | \sigma \in \gamma(c) \wedge ((c,\sigma_1) \in e \wedge (c,\sigma_2) \in e \Rightarrow \sigma_1 = \sigma_2)\}\} \cup \{\emptyset\}$$

eine Menge von Ein-/Ausgabe-Ereignissen und S eine endliche Menge von Zuständen ist und

$$M_\Omega := \{ (S, E_\Omega, \delta, \beta, s_0) | \delta : R \rightarrow S \wedge \beta : R \rightarrow E_\Omega \wedge R \subset S \times E_\Omega$$

$$\wedge (\forall [(s,x), y] \in \beta \, \forall (c_x, \sigma_x) \in x \, \forall (c_y, \sigma_y) \in y : c_x \neq c_y) \wedge s_0 \in S \},$$

B mit $B \subseteq C$ eine Menge von internen Synchronisationskanälen ist und die Komposition $comp_B : M_\Omega^n \rightarrow 2^{M_\Omega}$ von sequentiellen Maschinen definiert wird als

$$comp_B := \Big\{ \Big((K_1, ..., K_n), \widetilde{K} \Big) \Big|$$

$$(K_1, ..., K_n) = ((S_1, E_\Omega, \delta_1, \beta_1, s_{0_1}), ..., (S_n, E_\Omega, \delta_n, \beta_n, s_{0_n}))$$

$$\wedge \exists T = \{ \; ((x_1, ..., x_n), (y_1, ..., y_n), (s'_1, ..., s'_n), \widetilde{x}, \widetilde{y}) \mid$$

$$([(s_{0_1}, x_1), s'_1], ..., [(s_{0_n}, x_n), s'_n]) \in \delta_1 \times ... \times \delta_n$$

$$\wedge ([(s_{0_1}, x_1), y_1], ..., [(s_{0_n}, x_n), y_n]) \in \beta_1 \times ... \times \beta_n$$

$$\wedge \exists H_x = \bigcup_{i \in \{1,...,n\}} x_i \; \exists H_y = \bigcup_{i \in \{1,...,n\}} \beta_i(x_i) :$$

$$H_x \in E_\Omega \wedge H_y \in E_\Omega$$

$$\wedge \forall (c, \sigma) : \; (c \in B \Leftrightarrow (c, \sigma) \in H_x \cap H_y)$$

$$\wedge \widetilde{x} = H_x \setminus H_y \wedge \widetilde{y} = H_y \setminus H_x \; \}$$

$$\exists \widetilde{M}'_\Omega = \{ \; \widetilde{K}' \mid \exists ((x_1, ..., x_n), (y_1, ..., y_n), (s'_1, ..., s'_n), \widetilde{x}, \widetilde{y}) \in T :$$

$$\widetilde{K}' = comp_B ([(S_1, E_\Omega, \delta_1, \beta_1, s'_1), ..., (S_n, E_\Omega, \delta_n, \beta_n, s'_n)]) \; \} :$$

$$\widetilde{K} = \Big(\widetilde{S}, E_\Omega, \widetilde{\delta}, \widetilde{\beta}, \widetilde{s_0} \Big)$$

$$\wedge \widetilde{S} = (s_{0_1}, ..., s_{0_n}) \cup \bigcup_{\left(\widetilde{S'}, E'_\Omega, \widetilde{\delta'}, \widetilde{\beta'}, \widetilde{s'_0}\right) \in \widetilde{M}'_\Omega} \widetilde{S'}$$

$$\wedge \widetilde{\delta} = \{ \; [((s_{0_1}, ..., s_{0_n}), \widetilde{x}), (s'_1, ..., s'_n)] \mid$$

$$((x_1, ..., x_n), (y_1, ..., y_n), (s'_1, ..., s'_n), \widetilde{x}, \widetilde{y}) \in T \; \}$$

$$\cup \bigcup_{\left(\widetilde{S'}, E'_\Omega, \widetilde{\delta'}, \widetilde{\beta'}, \widetilde{s'_0}\right) \in \widetilde{M}'_\Omega} \widetilde{\delta'}$$

$$\wedge \widetilde{\beta} = \{ \; [((s_{0_1}, ..., s_{0_n}), \widetilde{x}), \widetilde{y}] \mid$$

$$((x_1, ..., x_n), (y_1, ..., y_n), (s'_1, ..., s'_n), \widetilde{x}, \widetilde{y}) \in T \; \}$$

$$\cup \bigcup_{\left(\widetilde{S'}, E'_\Omega, \widetilde{\delta'}, \widetilde{\beta'}, \widetilde{s'_0}\right) \in \widetilde{M}'_\Omega} \widetilde{\beta'}$$

$$\wedge \widetilde{s_0} = (s_{0_1}, ..., s_{0_n}) \; \Big\}.$$

12. Verfahren nach Anspruch 9, dadurch gekennzeichnet, dass die datenverarbeitende Netze durch eine Übersetzung von Algorithmen gebildet werden.

13. Verfahren nach einem der Ansprüche 9 bis 12, dadurch gekennzeichnet, dass wenigstens eine Komponente eine kryptologische Komponente ist.

14. Verfahren nach Anspruch 13, dadurch gekennzeichnet, dass wenigstens eine Komponente komprimierte Daten dekomprimiert.

15. Verfahren nach einem der Ansprüche 13 oder 14, dadurch gekennzeichnet, dass eine Komponente Daten liest und durch Veränderung dieser Daten ein für die Nutzung dieser Daten nicht oder wenig behinderndes Merkmal oder Wasserzeichen den Daten hinzufügt.

16. Verfahren nach Anspruch 13, dadurch gekennzeichnet, dass ein Verschlüsseler und ein Kombinierer mehrerer Eingabekanäle zu einem Ausgabekanal komponiert werden, wobei der Verschlüsseler und der Kombinierer Petri-Netze sind, der Kombinierer die über die Eingabekanäle empfangenen Daten auf den Ausgabekanal abbildet und die Ausgabe des Kombinierers die Eingabe für den Verschlüsseler ist.

17. Verfahren nach Anspruch 13, dadurch gekennzeichnet, dass ein Entschlüsseler und ein Separator komponiert werden, wobei der Entschlüsseler bzw. der Separator ein Petri-Netz ist und eine Umkehrung eines Verschlüsselers bzw. Kombinierers sein kann, der gemäß dem in Anspruch 16 beschriebenen Verfahren mit einem Kombinierer bzw. Verschlüsseler komponiert wurde, der Separator die über den Eingabekanal empfangenen Daten auf den Ausgabekanal abbildet und die Ausgabe des Entschlüsselers die Eingabe für den Separator ist.

18. Verfahren nach einem der Ansprüche 13 bis 17, dadurch gekennzeichnet, dass wenigstens eine Komponente eine kryptologische Komponente ist, die von einer geschützt ausgeführten kryptologischen Funktion Daten empfängt und verarbeitet, wobei das Kompositionsergebnis nicht oder fehlerhaft arbeitet, wenn von der kryptologischen Funktion keine oder fehlerhafte Daten empfangen werden.

19. Verfahren nach Anspruch 18, dadurch gekennzeichnet, dass eine Komposition unter Auslassung der kryptologischen Komponente oder wenigstens einer der kryptologischen Komponenten ausgeführt wird, wobei das Kompositionsergebnis bezüglich der Nutzbarkeit gegenüber dem Kompositionsergebnis, das ohne die Auslassung gebildet wurde, wenigsten eine Einschränkung besitzt.

20. Verfahren zur Verarbeitung von Daten, insbesondere in Verbindung mit einem der Ansprüche 1 bis 19, dadurch gekennzeichnet, dass ein datenverarbeitendes Netz bzw. ein Programm von einer geschützt ausgeführten kryptologischen Funktion zweite Daten empfängt und verarbeitet und das datenverarbeitende Netz bzw. das Programm nicht oder fehlerhaft arbeitet, wenn keine oder fehlerhafte zweite Daten empfangen werden, wobei die kryptologische Funktion mit der das datenverarbeitende Netz bzw. das Programm ausführenden Vorrichtung fest verbunden ist.

21. Verfahren nach Anspruch 20, dadurch gekennzeichnet, dass ein Wert über die Berechnung eines Funktionswertes der kryptologischen Funktion hinaus für einen Angreifer nicht lesbar oder veränderbar gespeichert wird und bei einer weiteren Berechnung eines weiteren Funktionswertes dieser Wert das Ergebnis der weiteren Berechnung beeinflusst, wobei der Wert sich nach einer vorgegebenen Regel verändert.

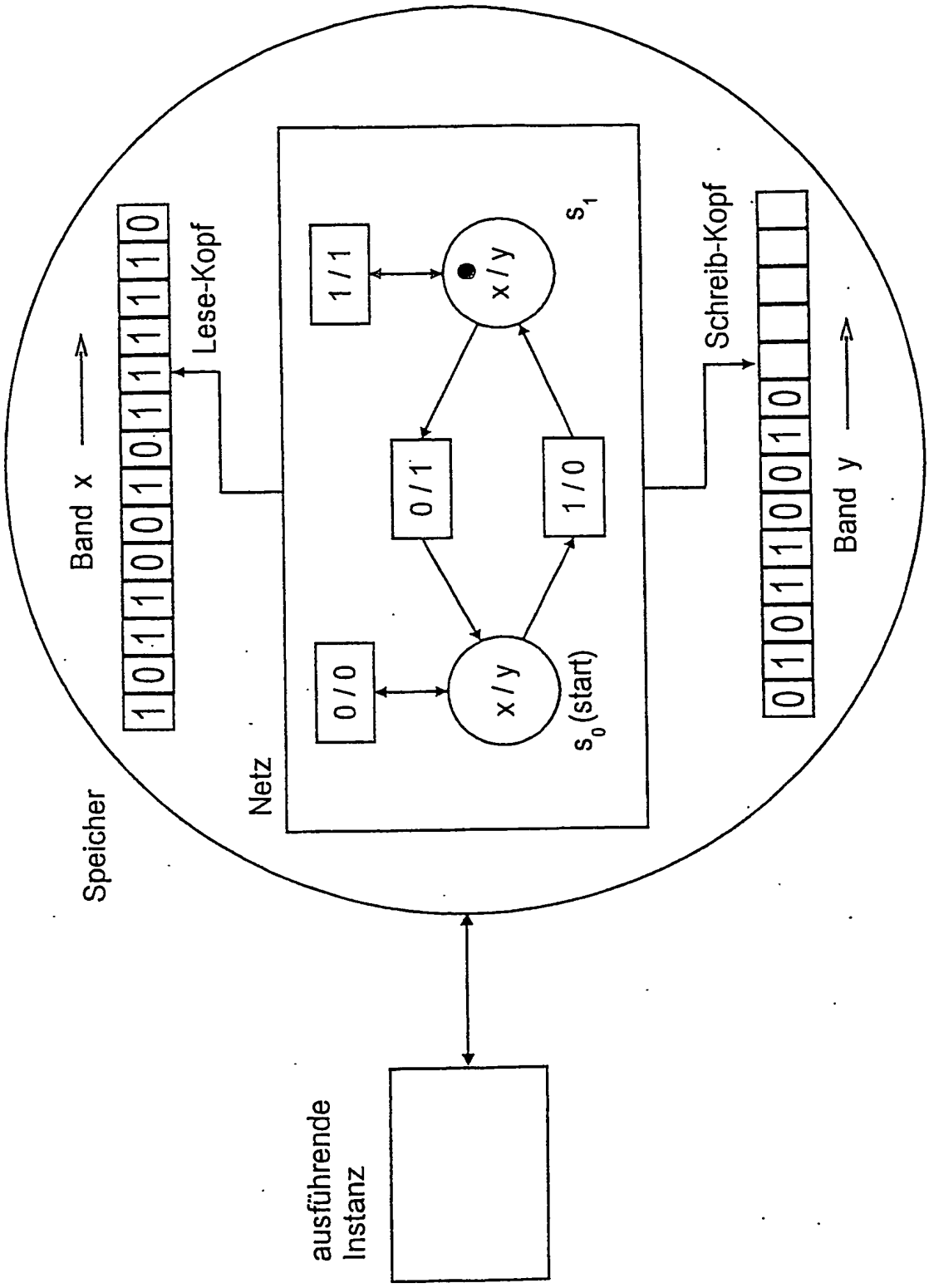

Speicher
Band x
1 0 1 1 0 0 1 0 1 1 1 1 1 0
Lese-Kopf
Netz
1 / 1
x / y
s_1
0 / 1
1 / 0
0 / 0
x / y
s_0(start)
Schreib-Kopf
0 1 0 1 1 0 0 1 0
Band y
ausführende Instanz

Fig. 1

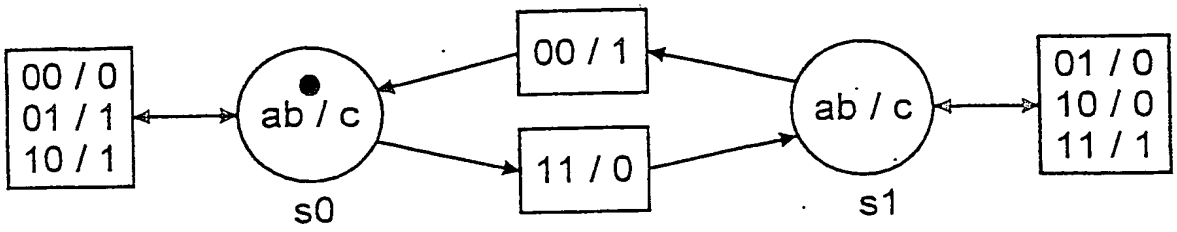

00 / 0
01 / 1
10 / 1
ab / c
s0
00 / 1
11 / 0
ab / c
s1
01 / 0
10 / 0
11 / 1

Fig. 2

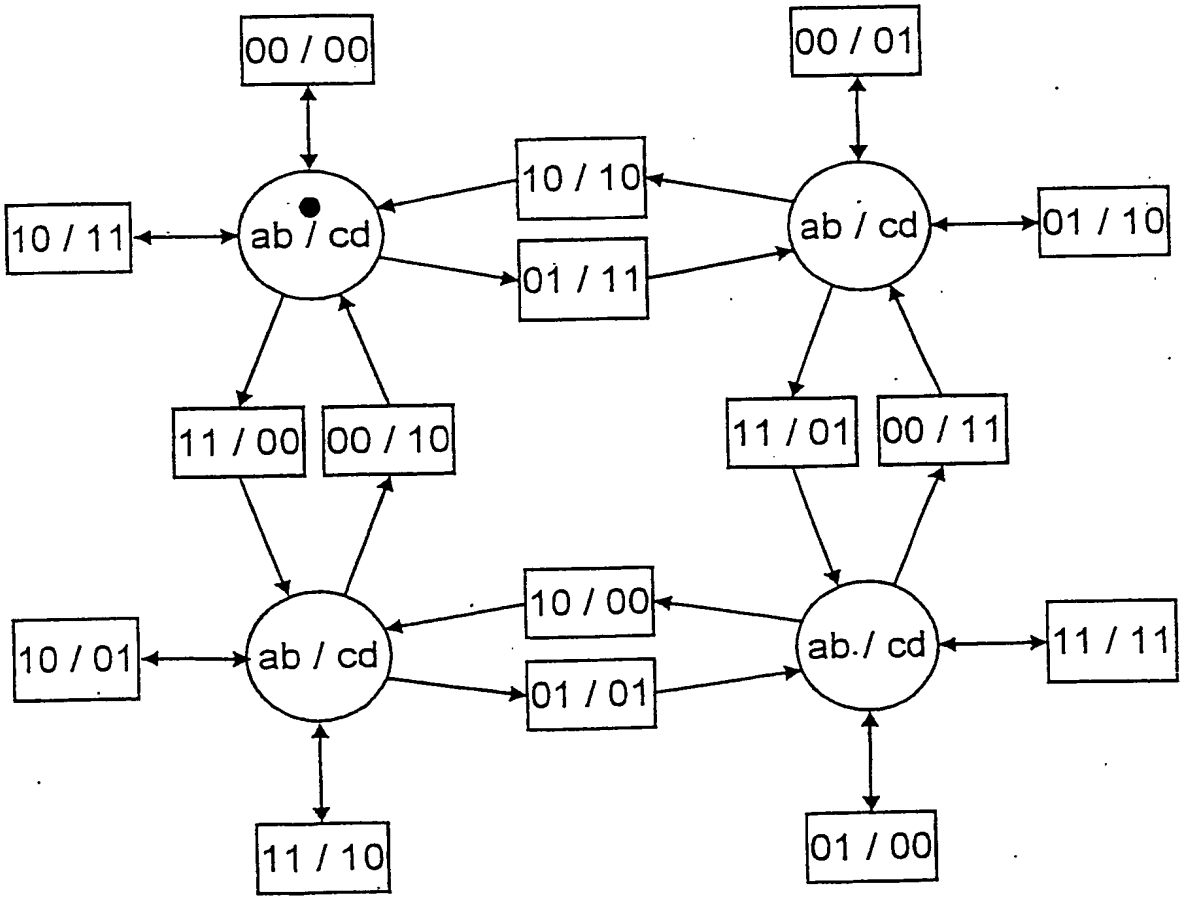

00 / 00
00 / 01
10 / 10
10 / 11
ab / cd
01 / 11
ab / cd
01 / 10
11 / 00
00 / 10
11 / 01
00 / 11
10 / 00
10 / 01
ab / cd
01 / 01
ab./ cd
11 / 11
11 / 10
01 / 00

Fig. 3

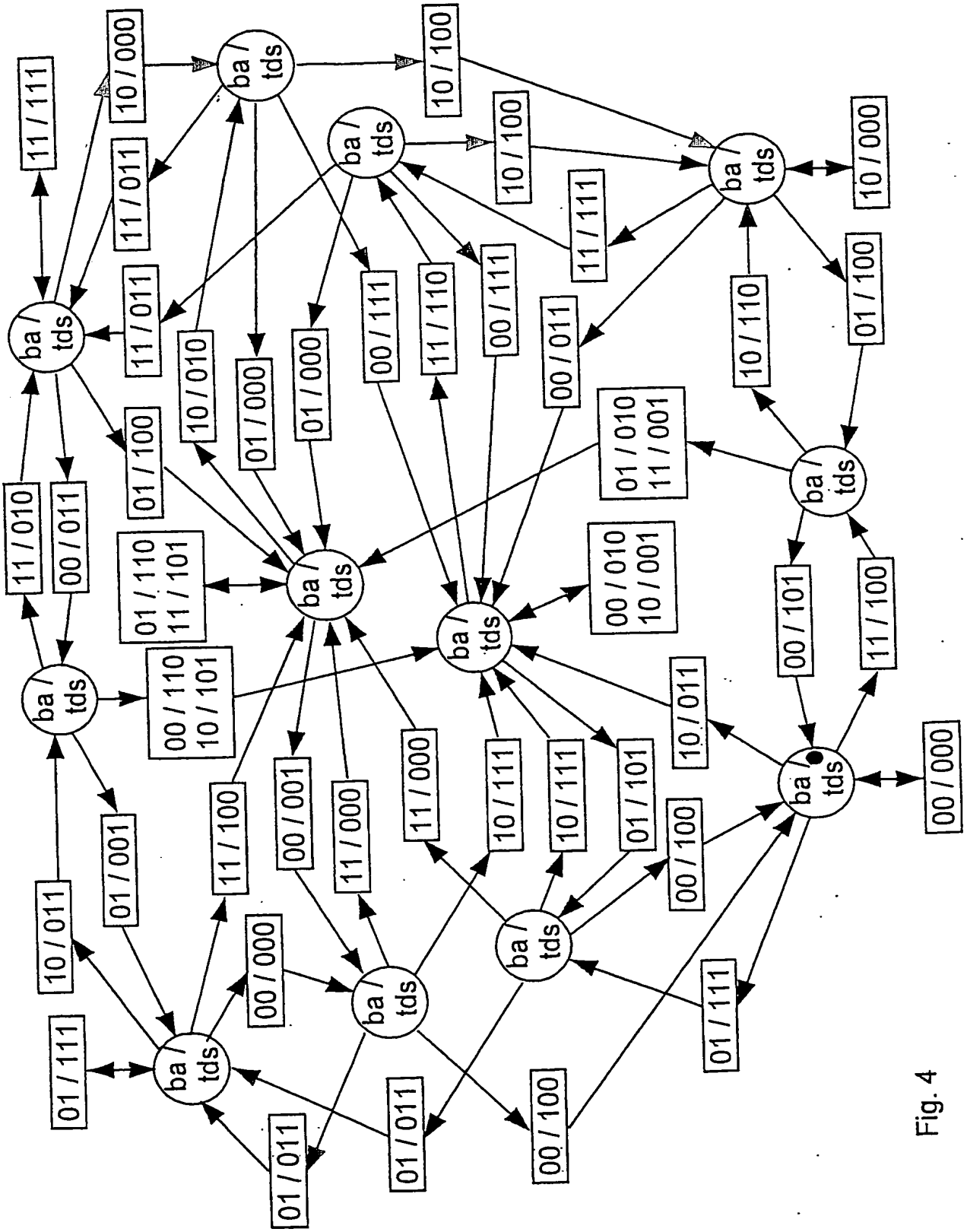

ba / tds
11 / 111
10 / 000
11 / 011
10 / 100
11 / 010
00 / 011
01 / 100
01 / 110
11 / 101
00 / 110
10 / 101
01 / 001
10 / 011
01 / 111
11 / 100
00 / 001
01 / 000
00 / 000
11 / 000
01 / 011
00 / 111
11 / 110
00 / 010
10 / 001
10 / 111
01 / 101
00 / 100
10 / 110
01 / 010
11 / 001
00 / 101
11 / 100
00 / 011
01 / 100
10 / 000
00 / 000

Fig. 4

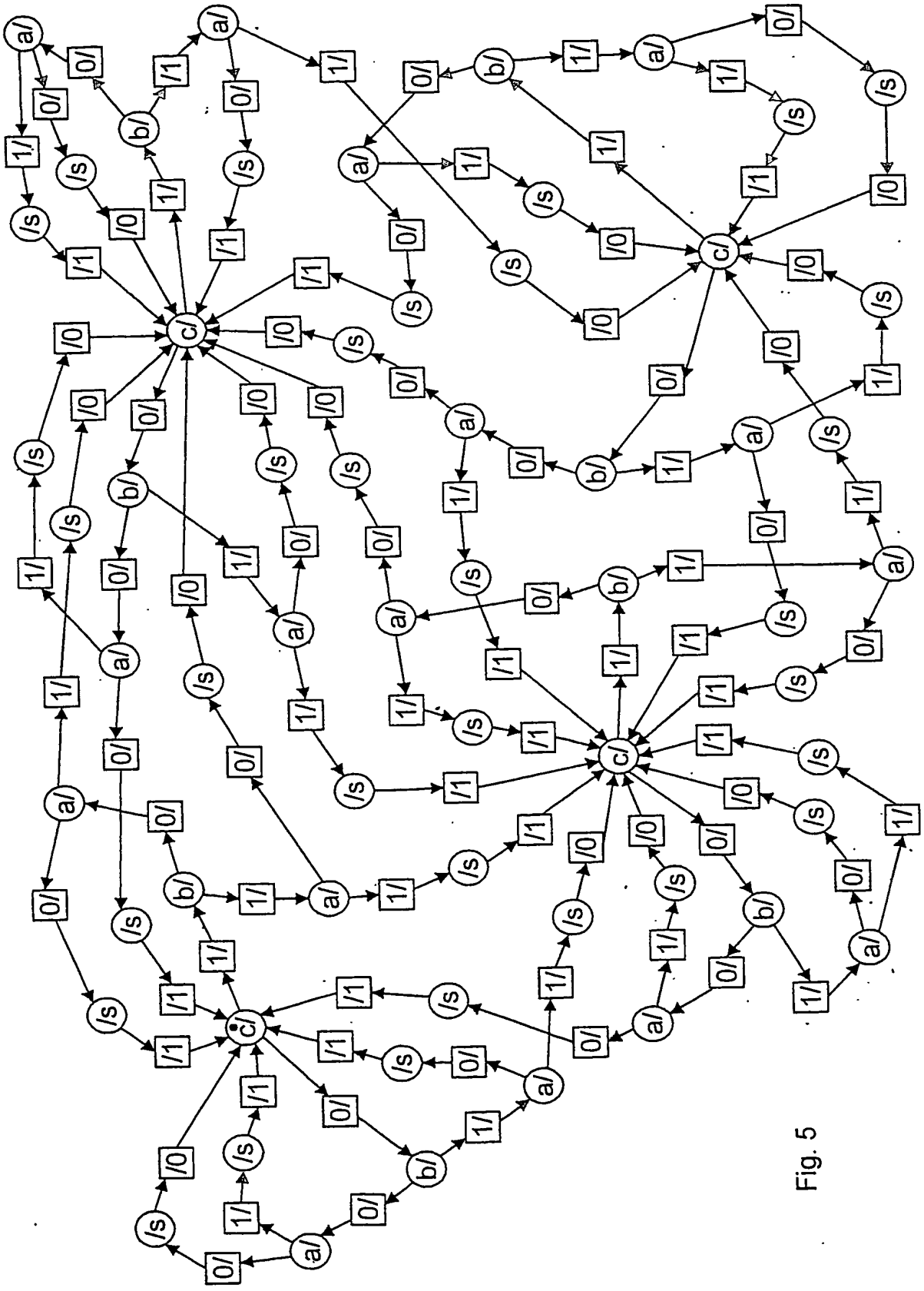


Fig. 5

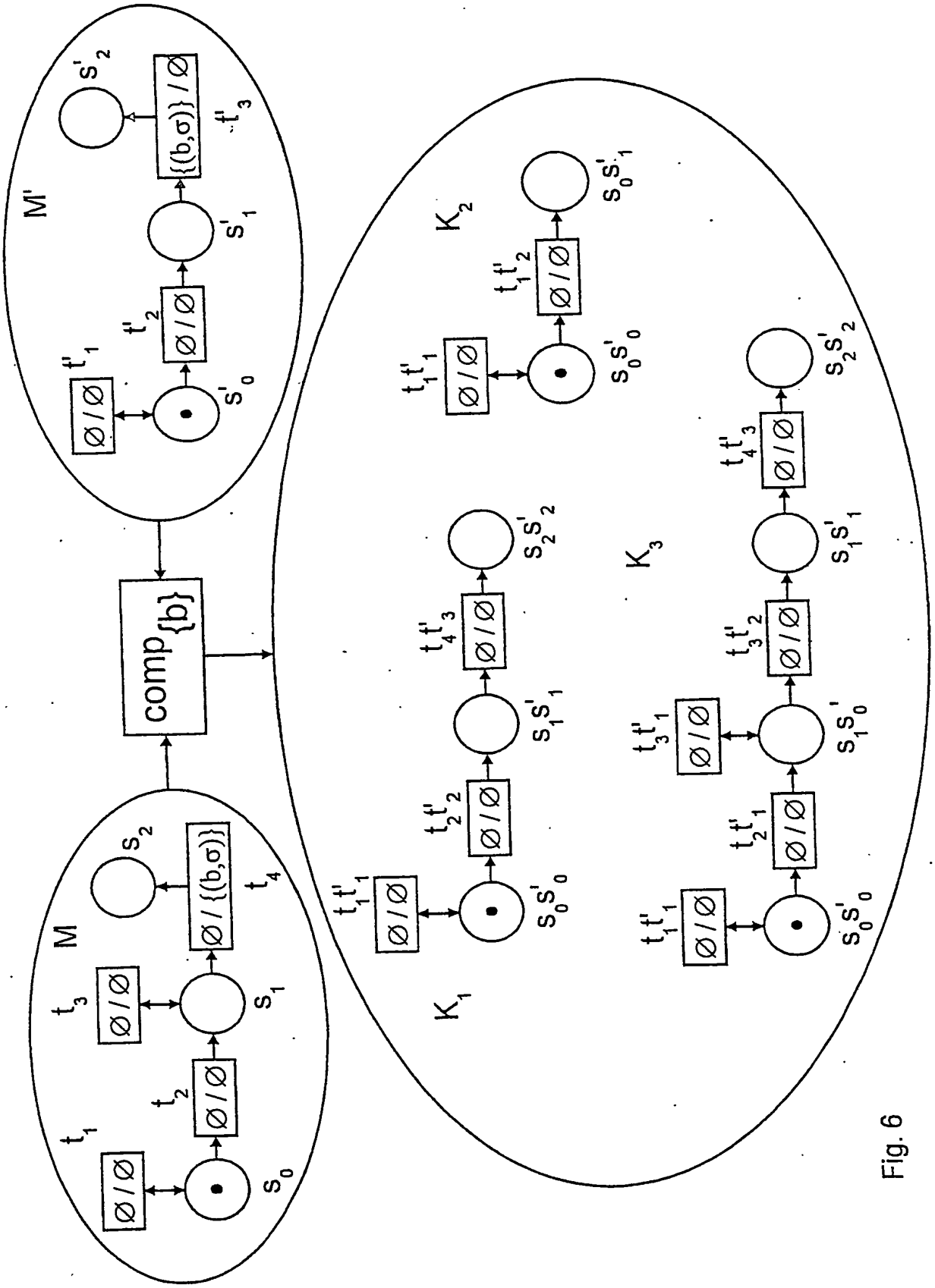


Fig. 6

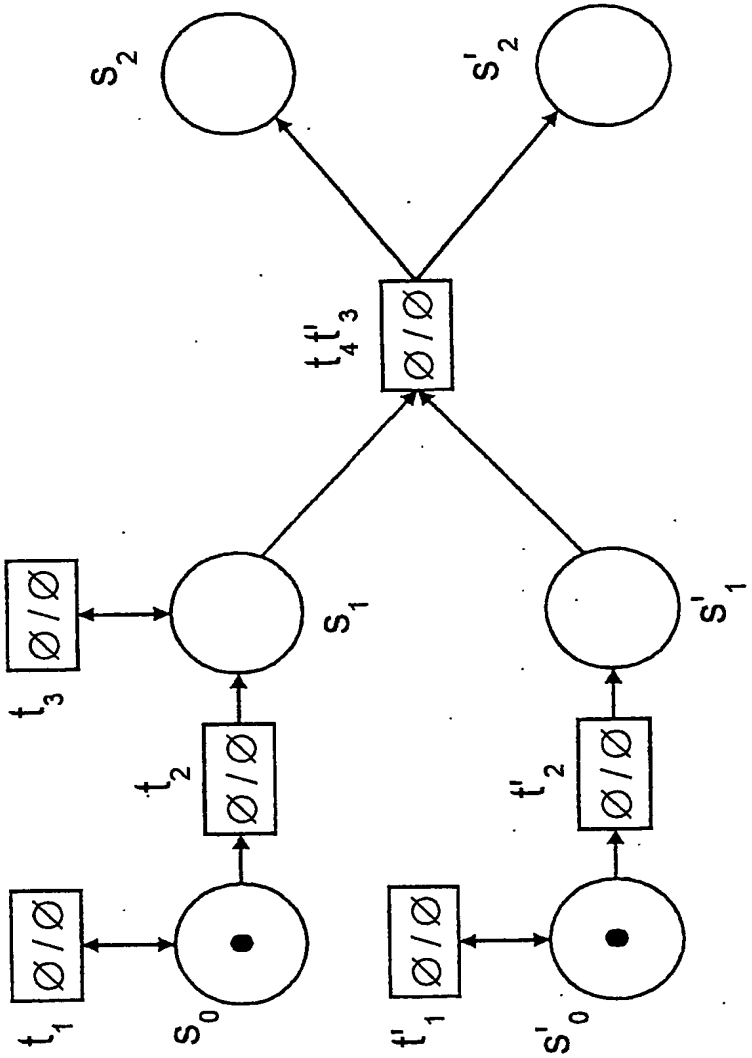


Fig. 7

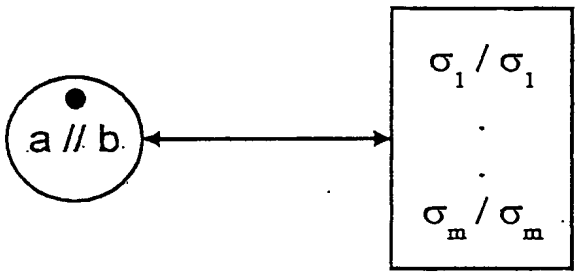


Fig. 8

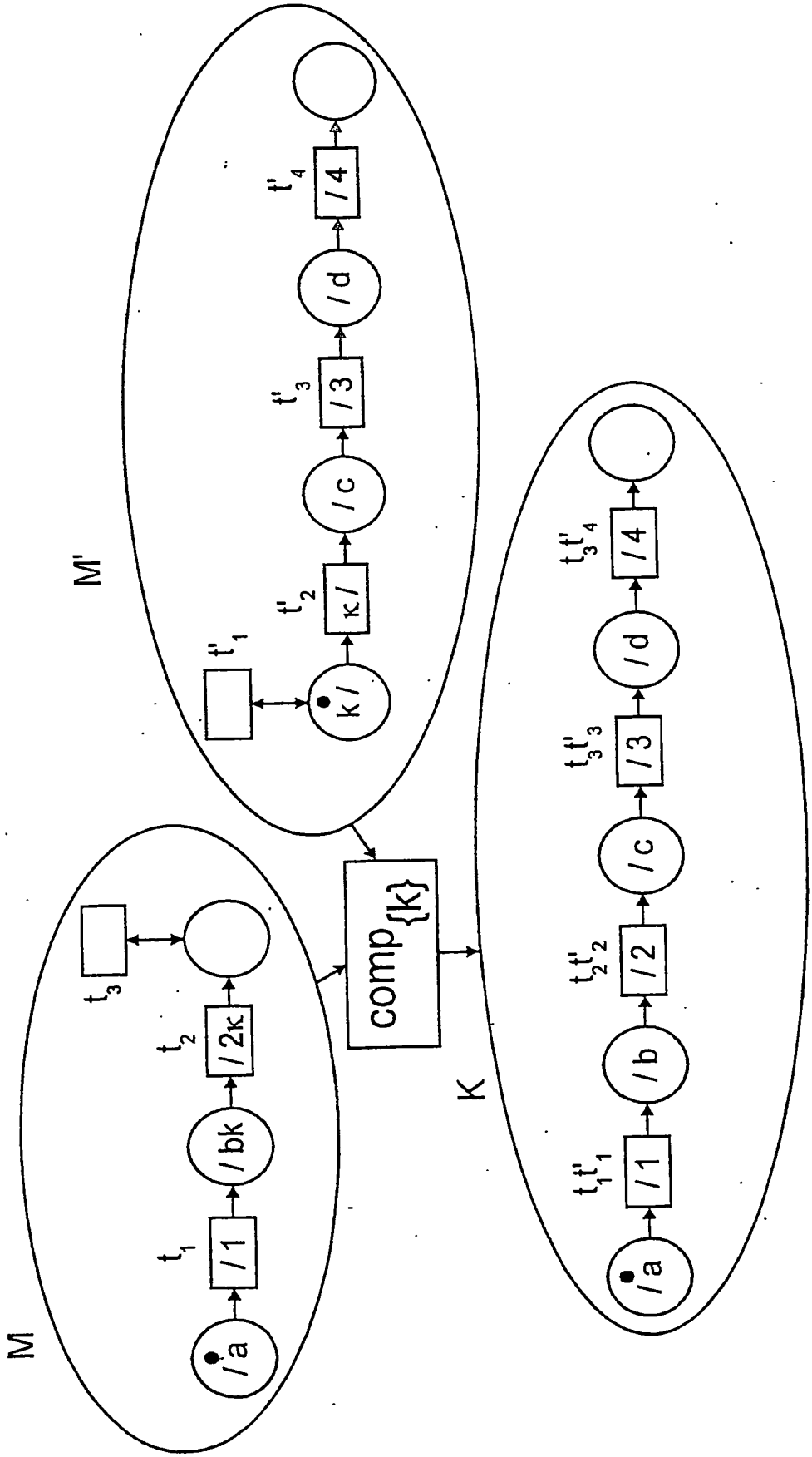
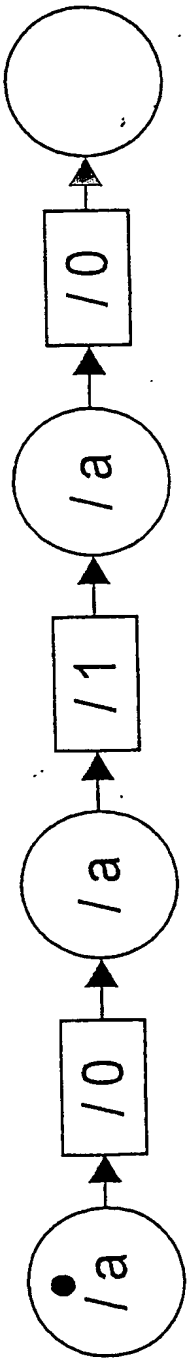

M
/a
t1
/1
/bk
t2
/2k
t3
M'
k/
t'1
t'2
κ/
/c
t'3
/3
/d
t'4
/4
comp{k}
K
/a
t1t'1
/1
/b
t2t'2
/2
/c
t3t'3
/3
/d
t3t'4
/4

Fig. 9



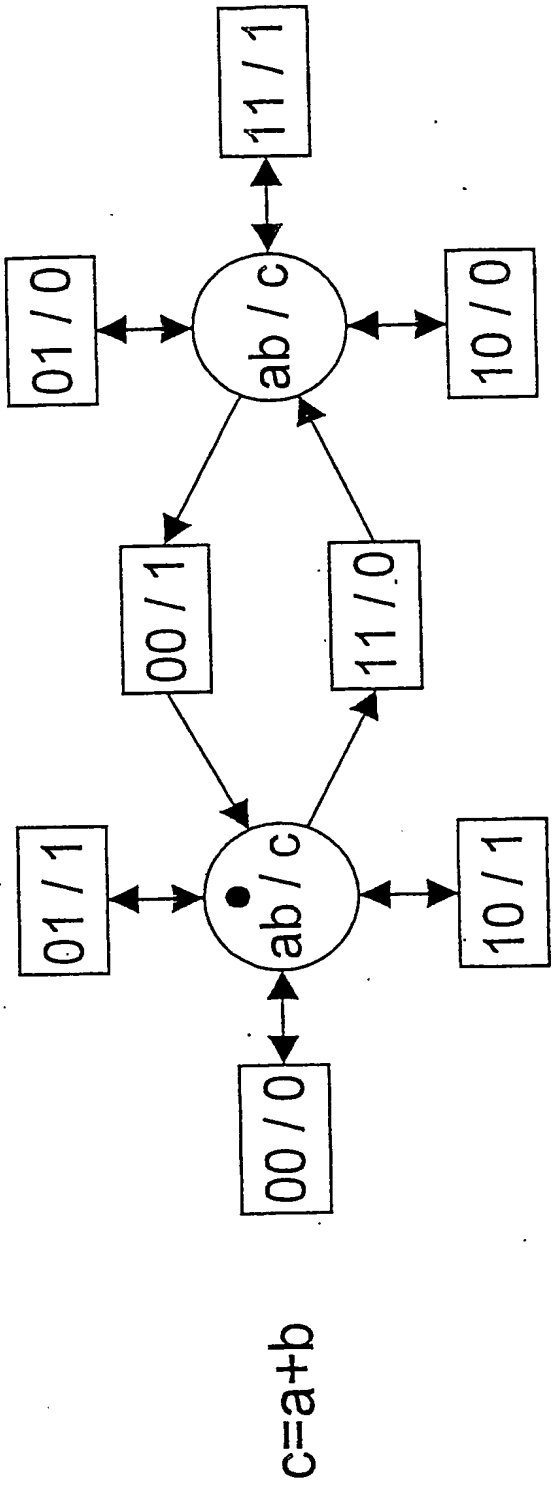


Fig.10

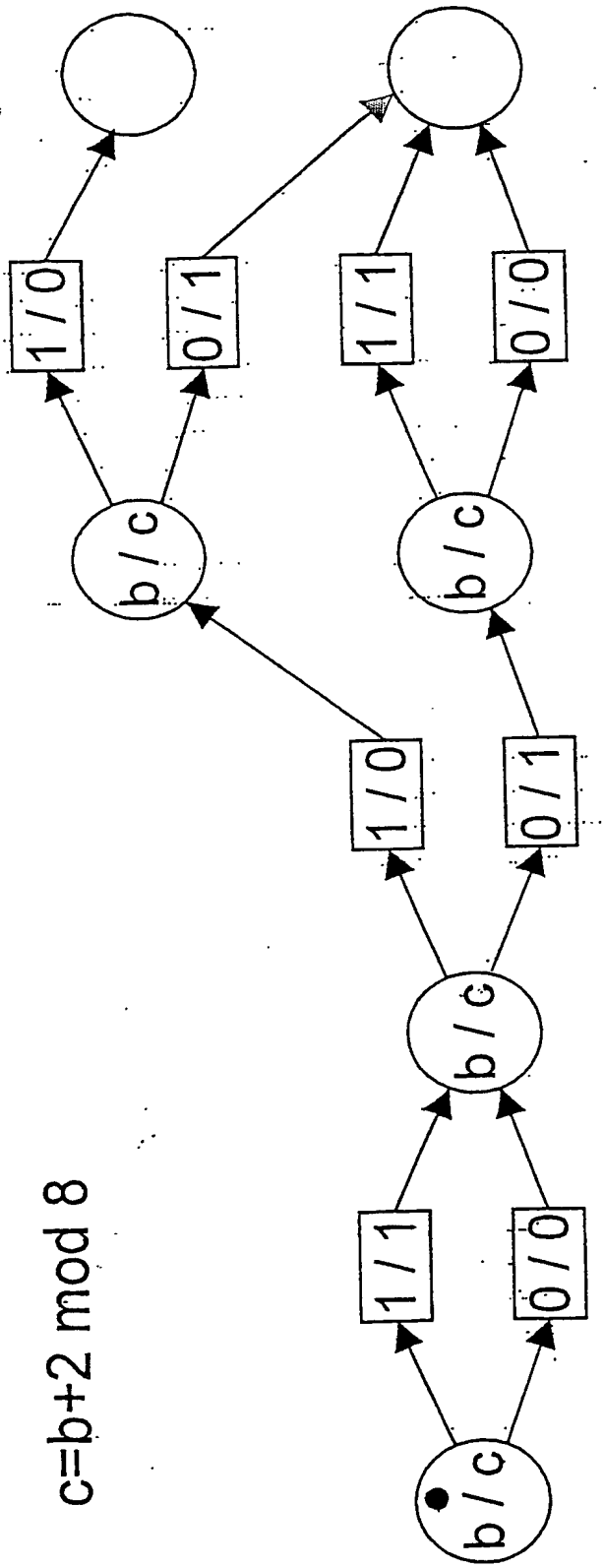


Fig. 11

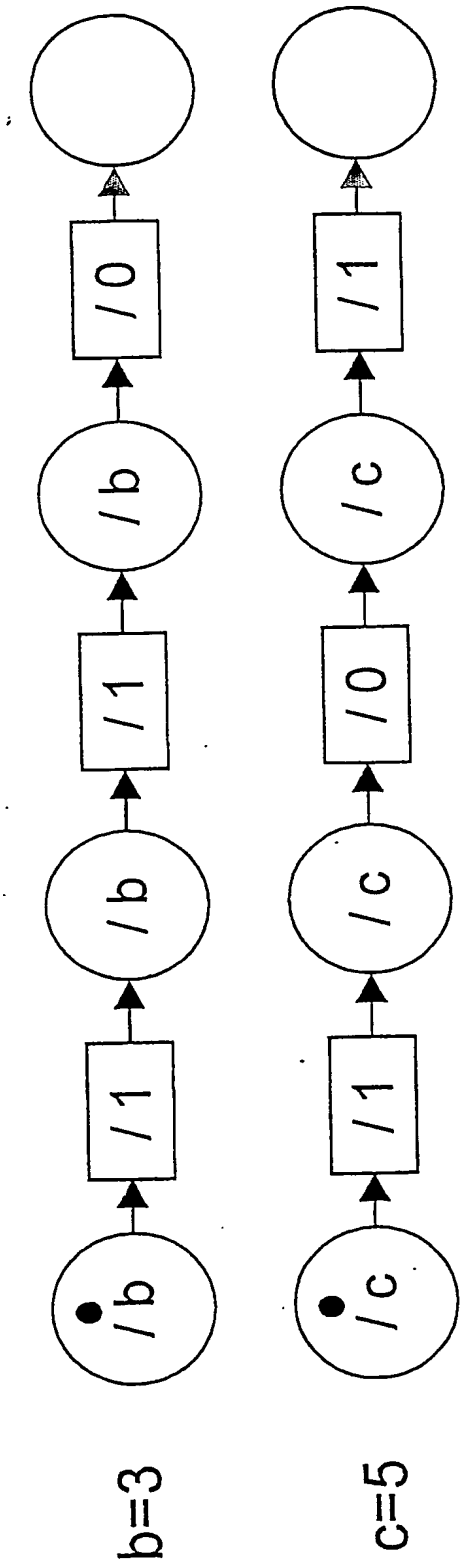


Fig. 12

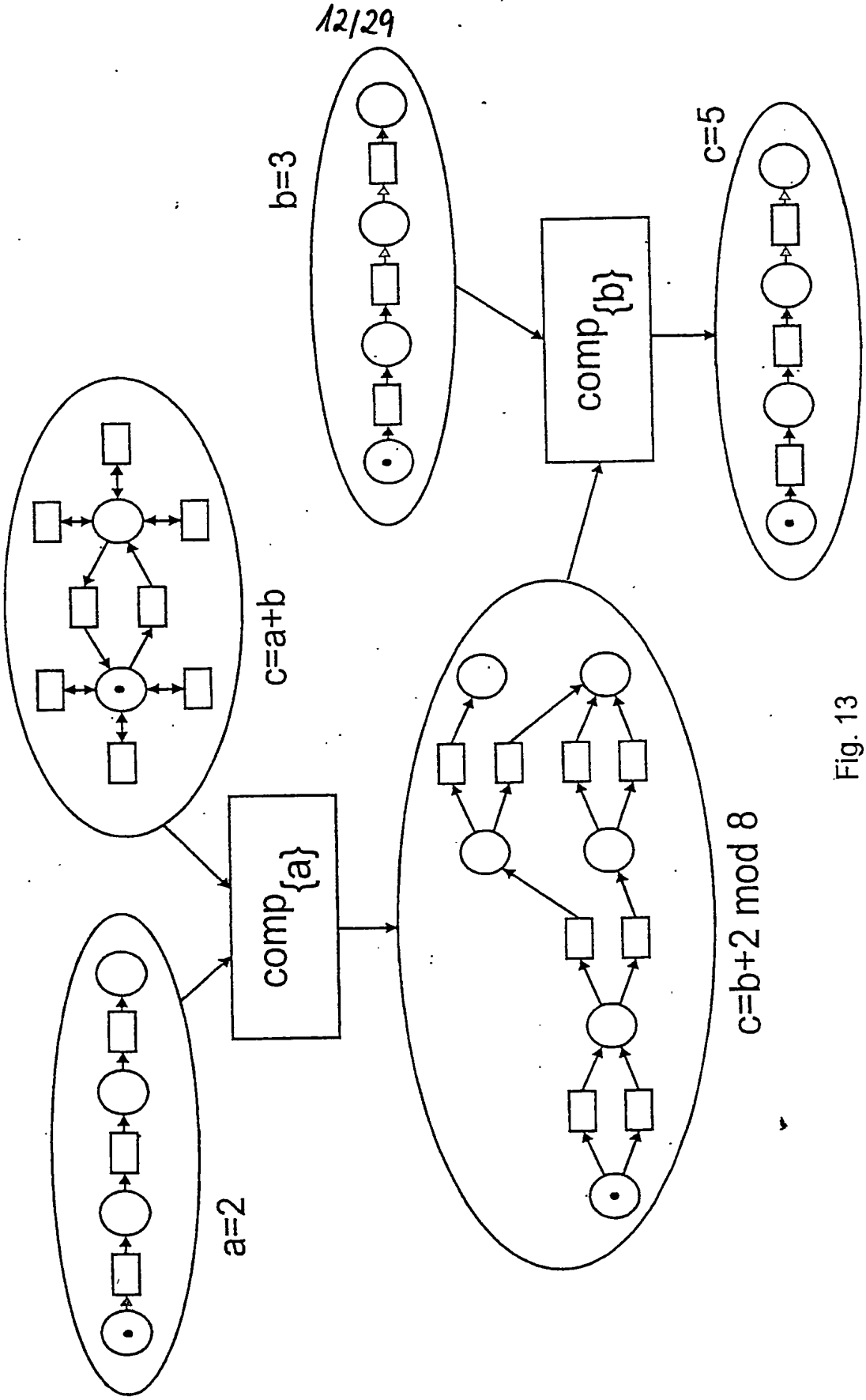


Fig. 13

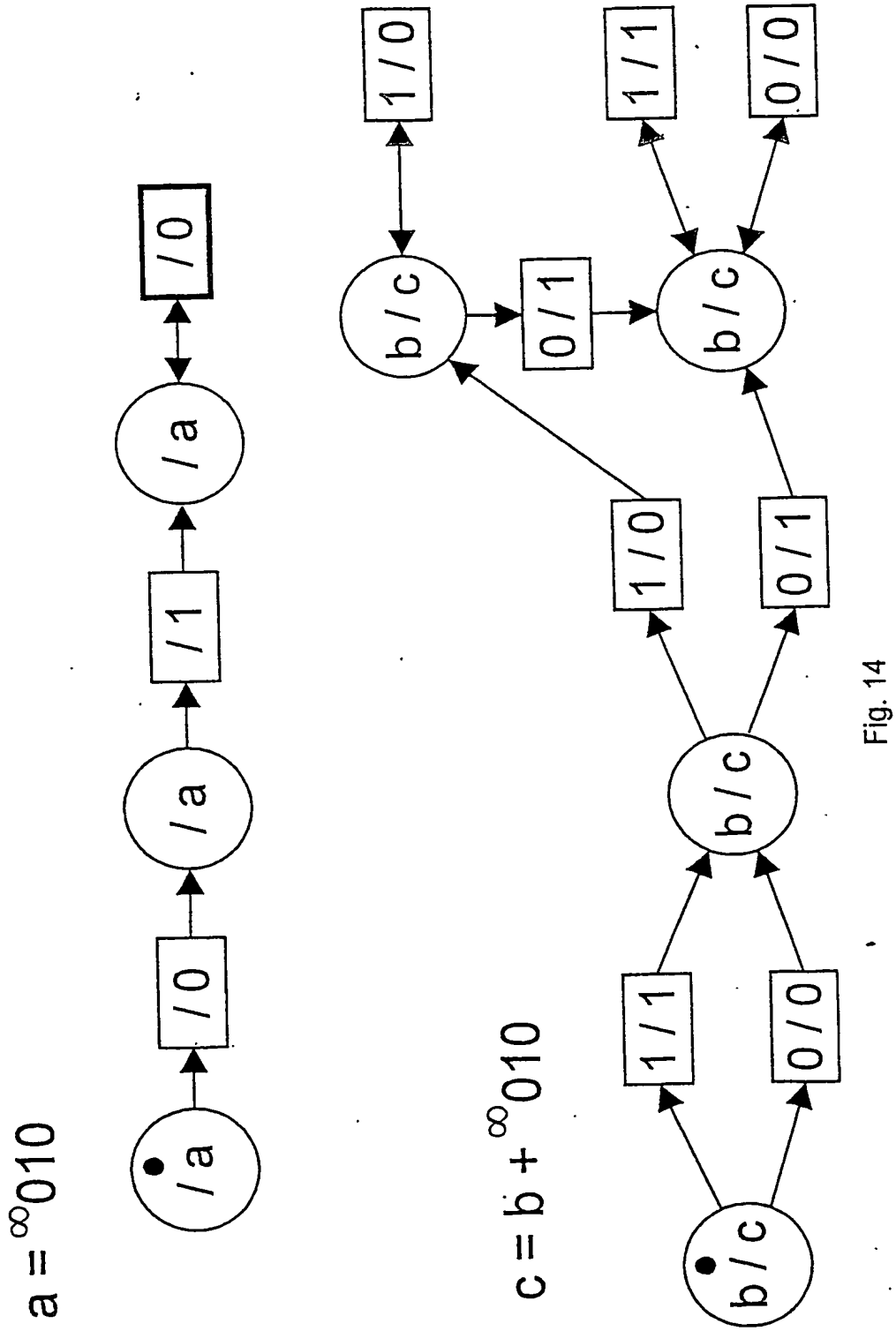


Fig. 14

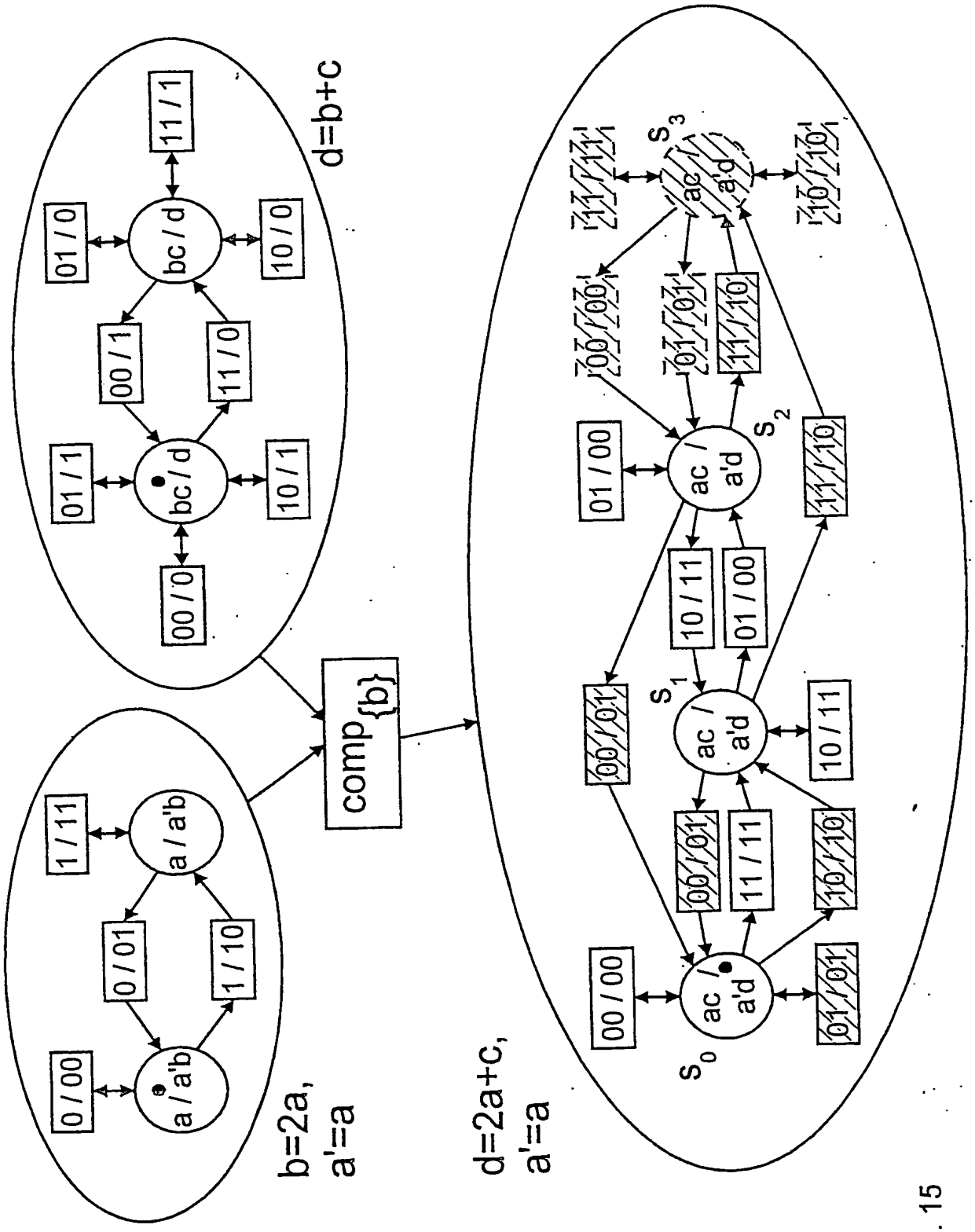

d=b+c
bc / d
bc / d
a / a'b
a / a'b
comp{b}
b=2a,
a'=a
d=2a+c,
a'=a
ac / a'd
ac / a'd
ac / a'd
ac / a'd
s0
s1
s2
s3
00 / 00
01 / 01
10 / 10
11 / 11
0 / 00
0 / 01
1 / 10
1 / 11
00 / 0
00 / 1
01 / 0
01 / 1
10 / 0
10 / 1
11 / 0
11 / 1
01 / 00
10 / 11
01 / 00
11 / 10
11 / 10

Fig. 15

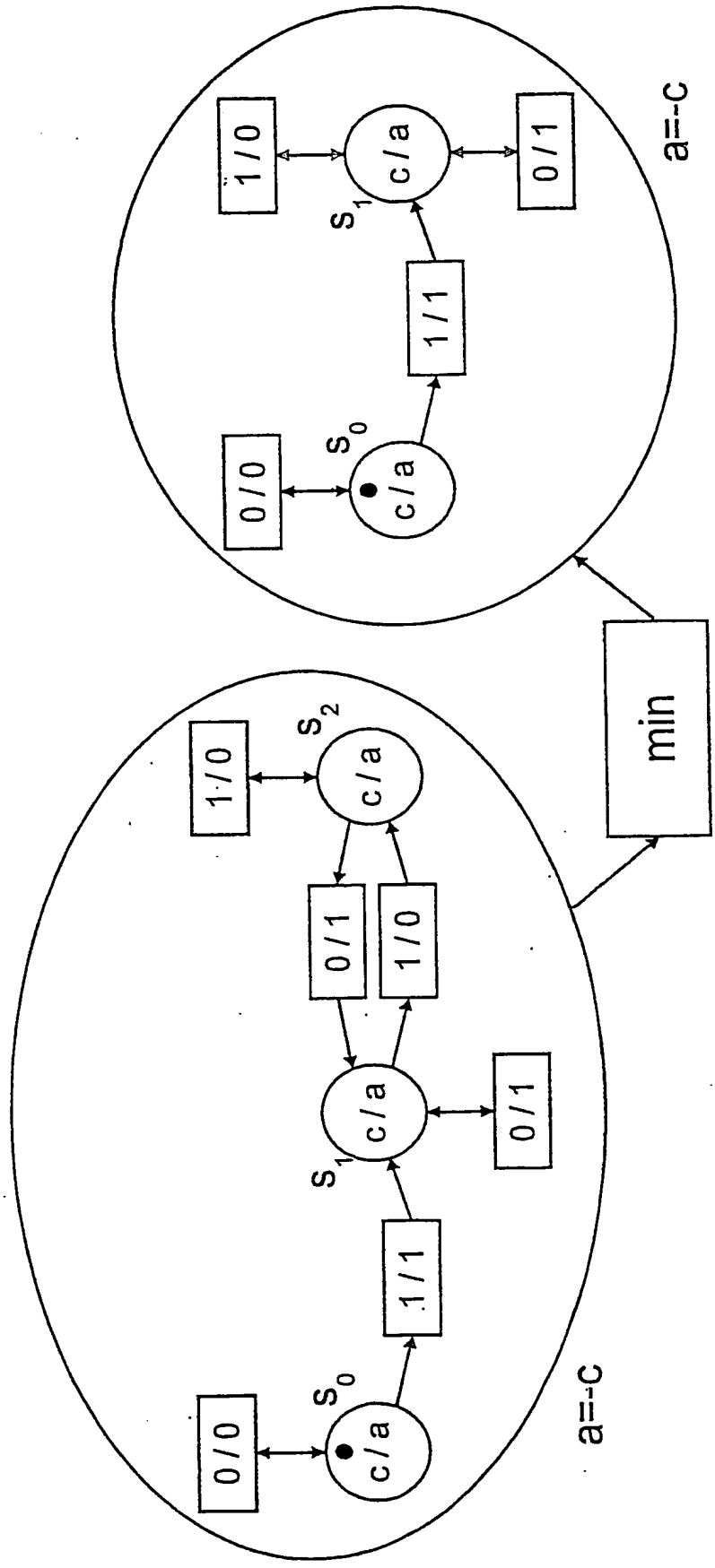


Fig. 16

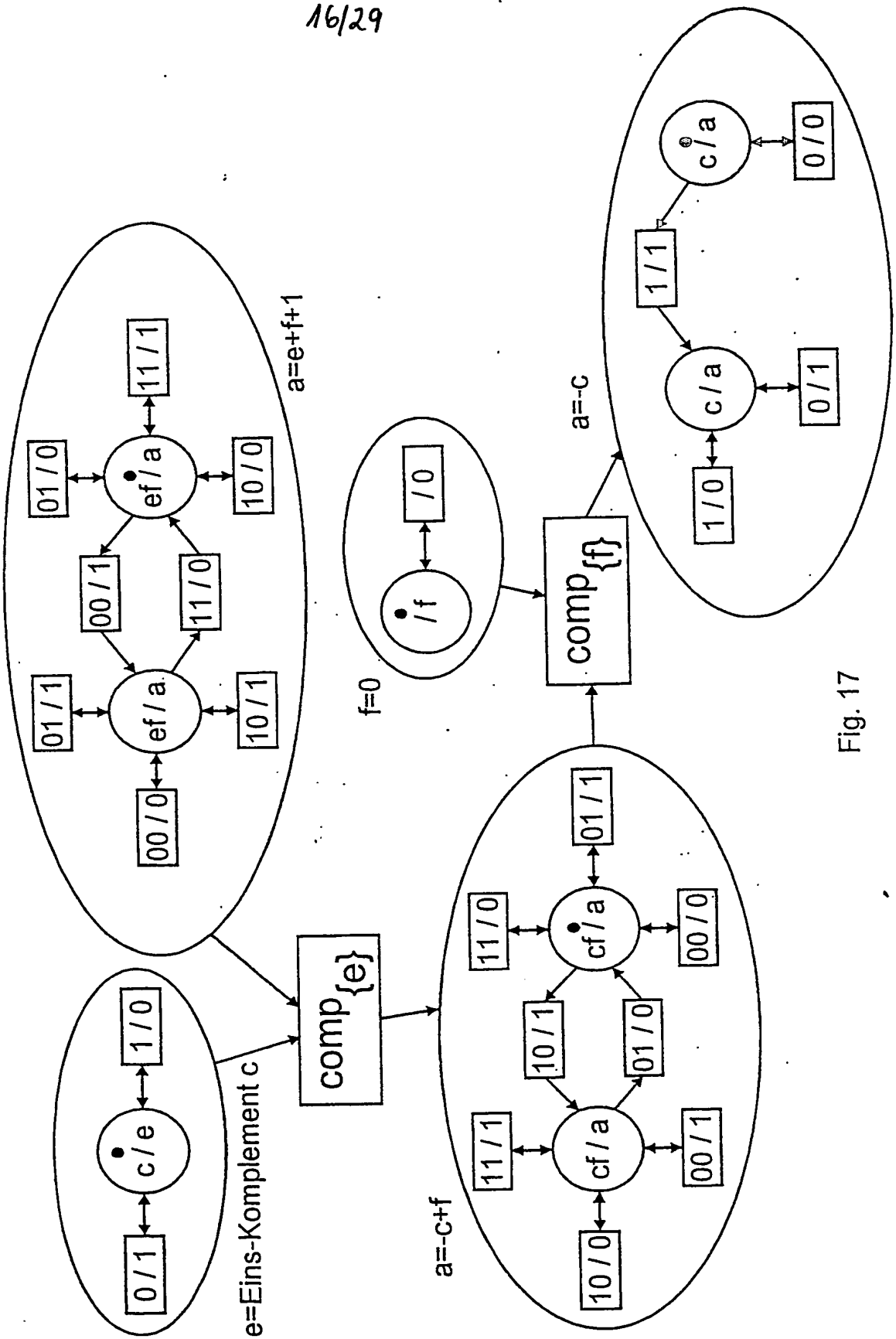

a=e+f+1
ef / a
11 / 1
01 / 0
10 / 0
00 / 1
11 / 0
01 / 1
ef / a
10 / 1
00 / 0
e=Eins-Komplement c
c / e
1 / 0
0 / 1
comp{e}
f=0
/ f
/ 0
comp{f}
a=-c
c / a
1 / 1
0 / 0
1 / 0
c / a
0 / 1
a=-c+f
cf / a
01 / 1
11 / 0
00 / 0
10 / 1
01 / 0
cf / a
11 / 1
00 / 1
10 / 0

Fig. 17

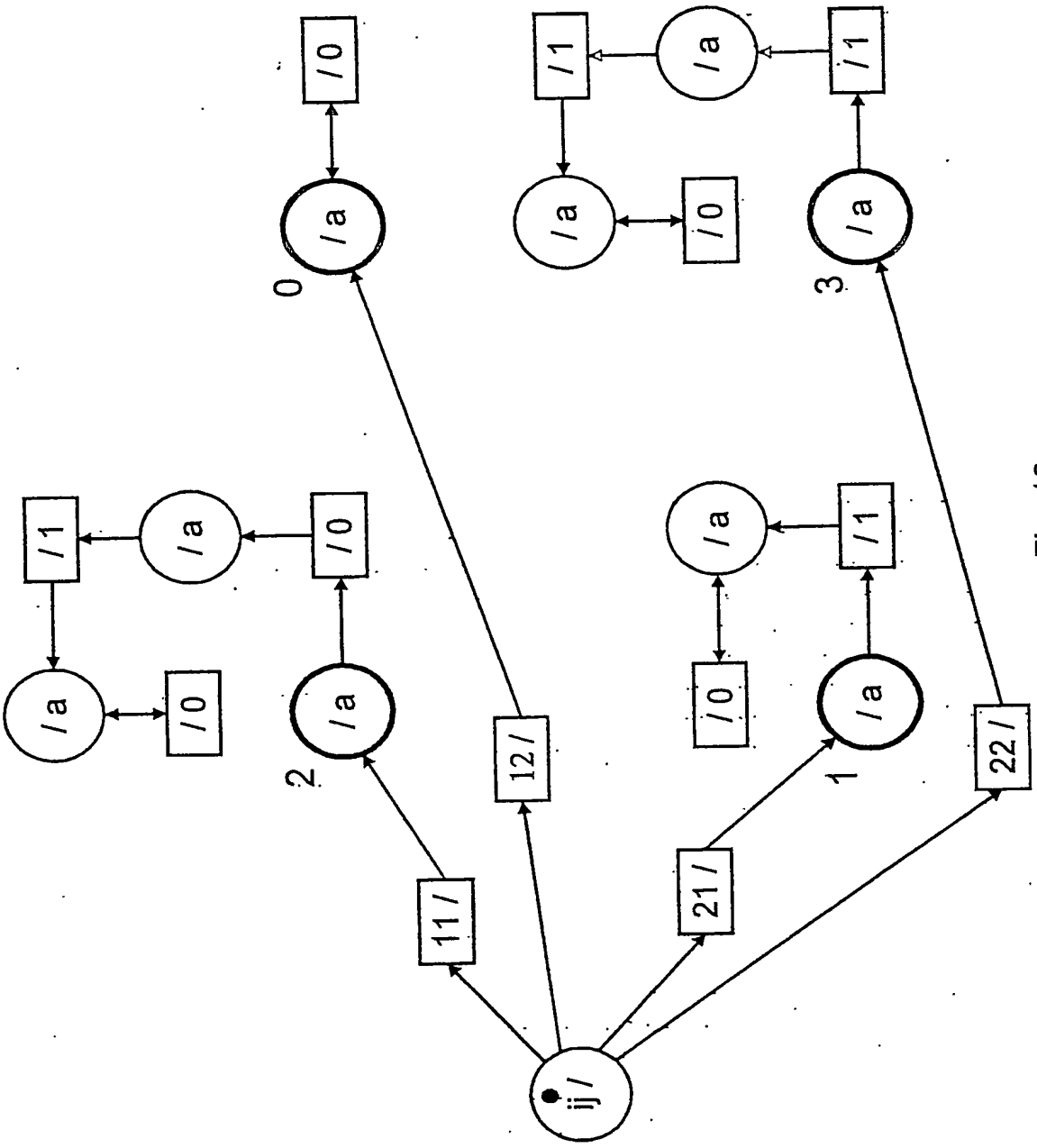


Fig. 18

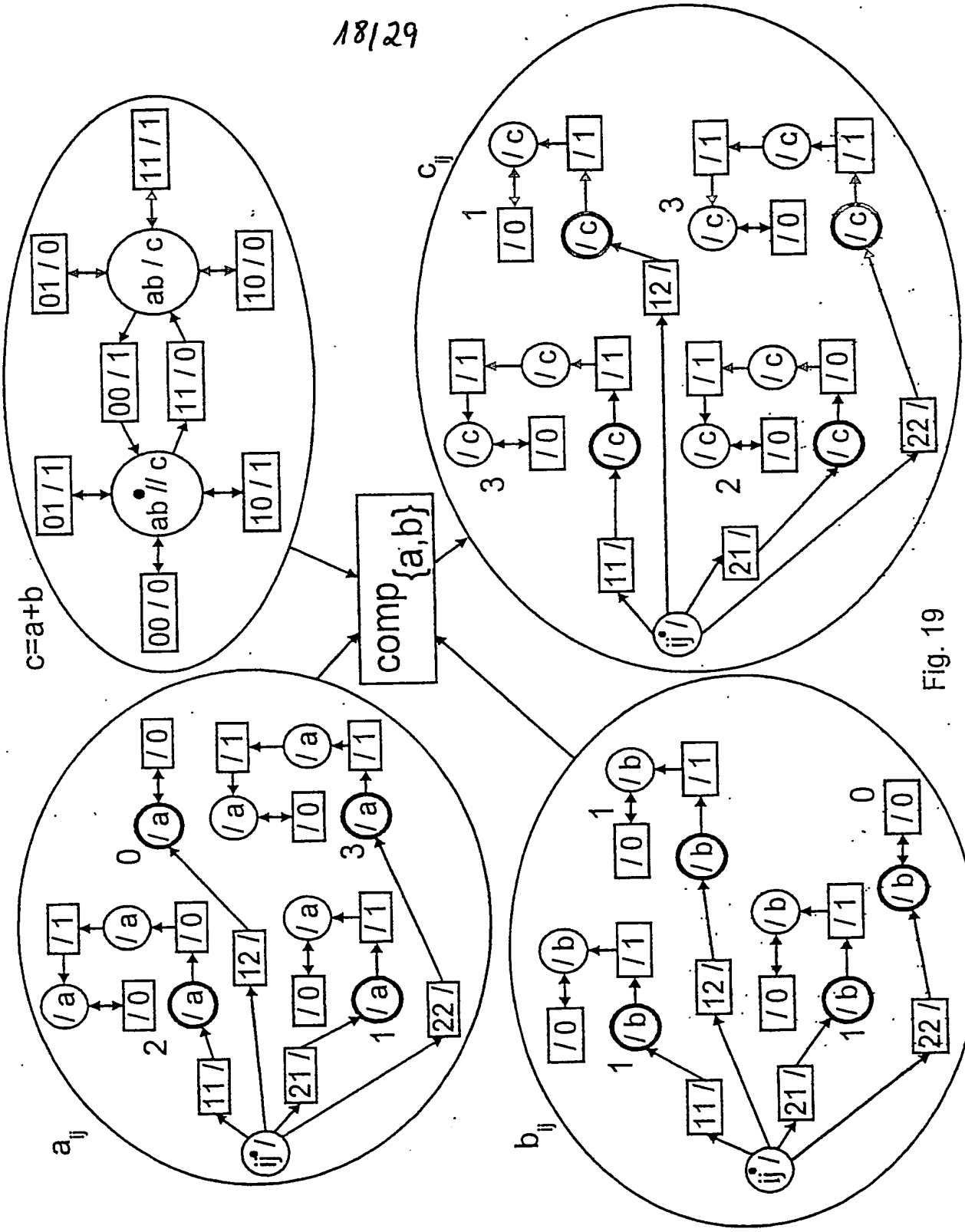


Fig. 19

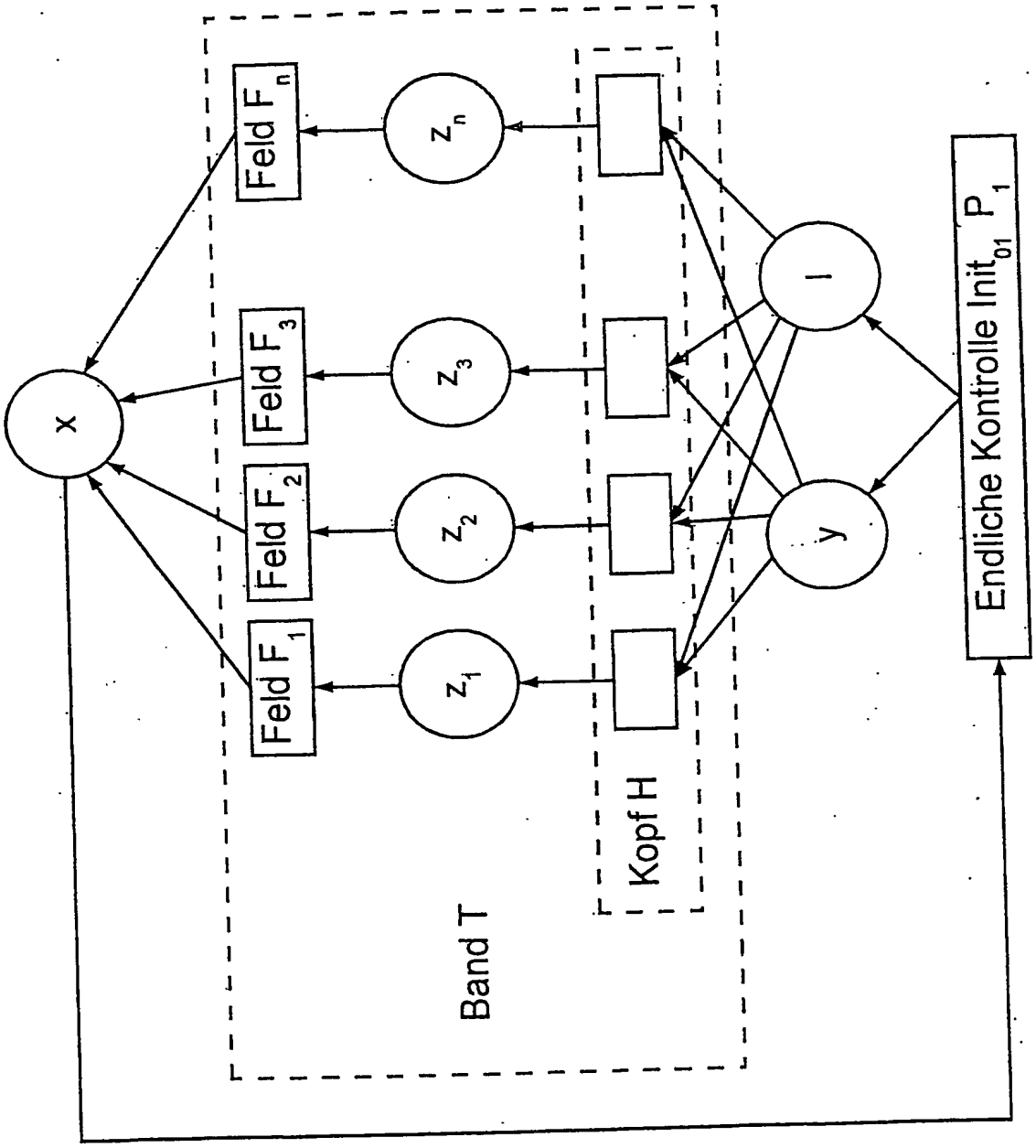

Feld F_n
Feld F_3
Feld F_2
Feld F_1
z_n
z_3
z_2
z_1
x
y
1
Endliche Kontrolle $Init_{01}$ P_1
Band T
Kopf H

Fig. 20

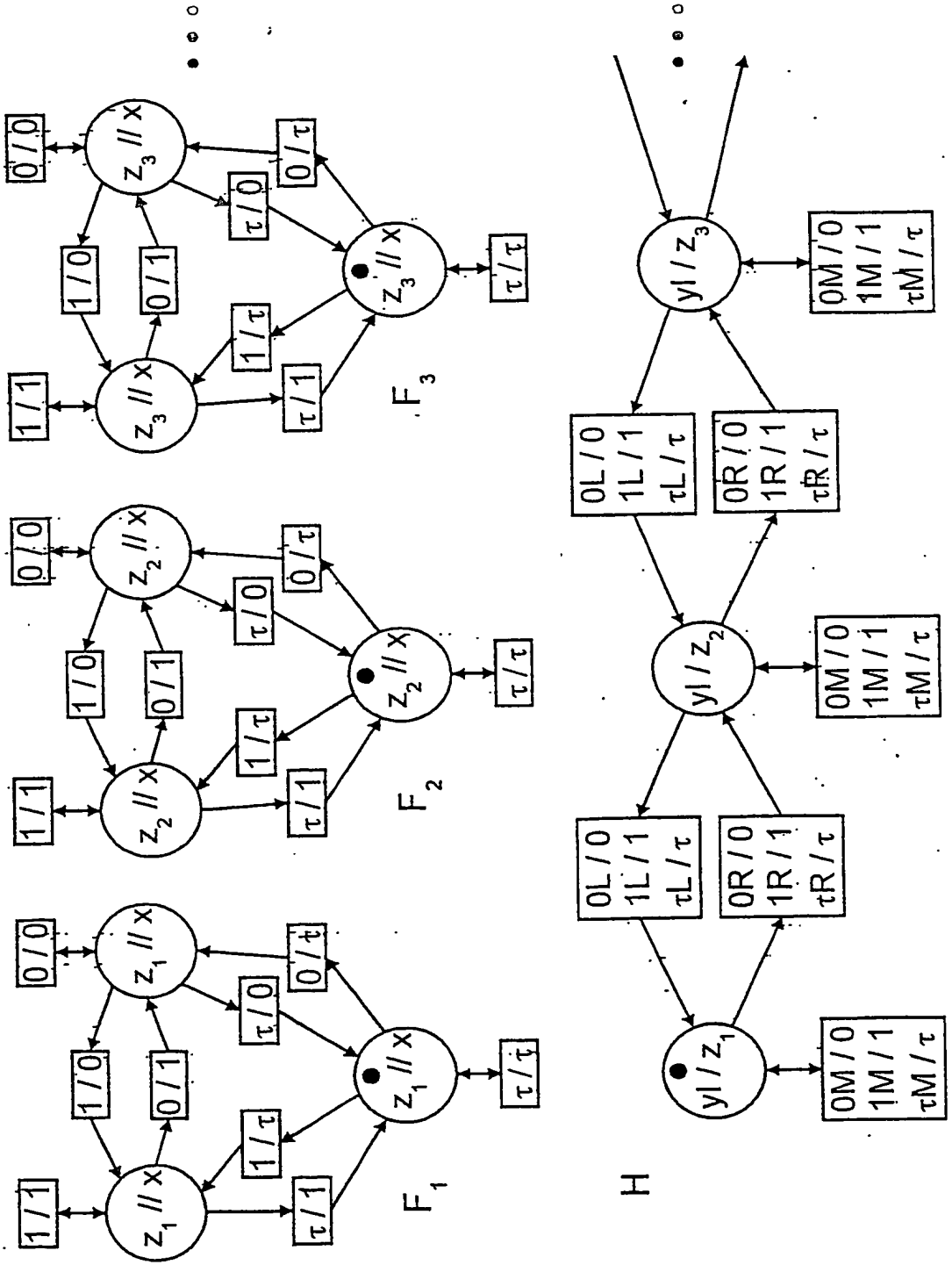

H
F1
F2
F3
yl / z1
yl / z2
yl / z3
0L / 0
1L / 1
τL / τ
0R / 0
1R / 1
τR / τ
0M / 0
1M / 1
τM / τ
z1 // x
z2 // x
z3 // x
0 / 0
1 / 0
0 / 1
1 / 1
τ / 0
0 / τ
1 / τ
τ / 1
τ / τ

Fig. 21

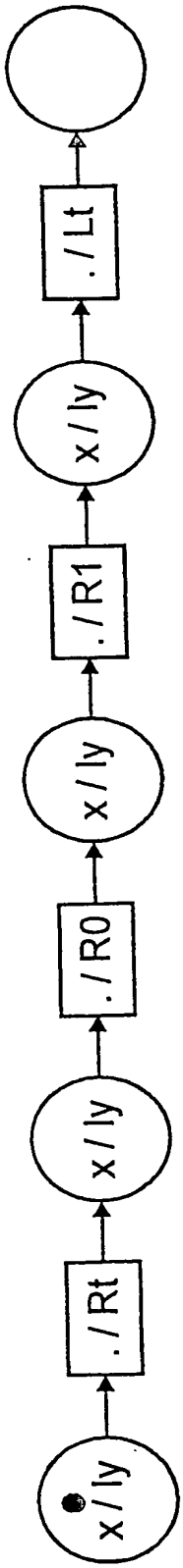


Fig.22

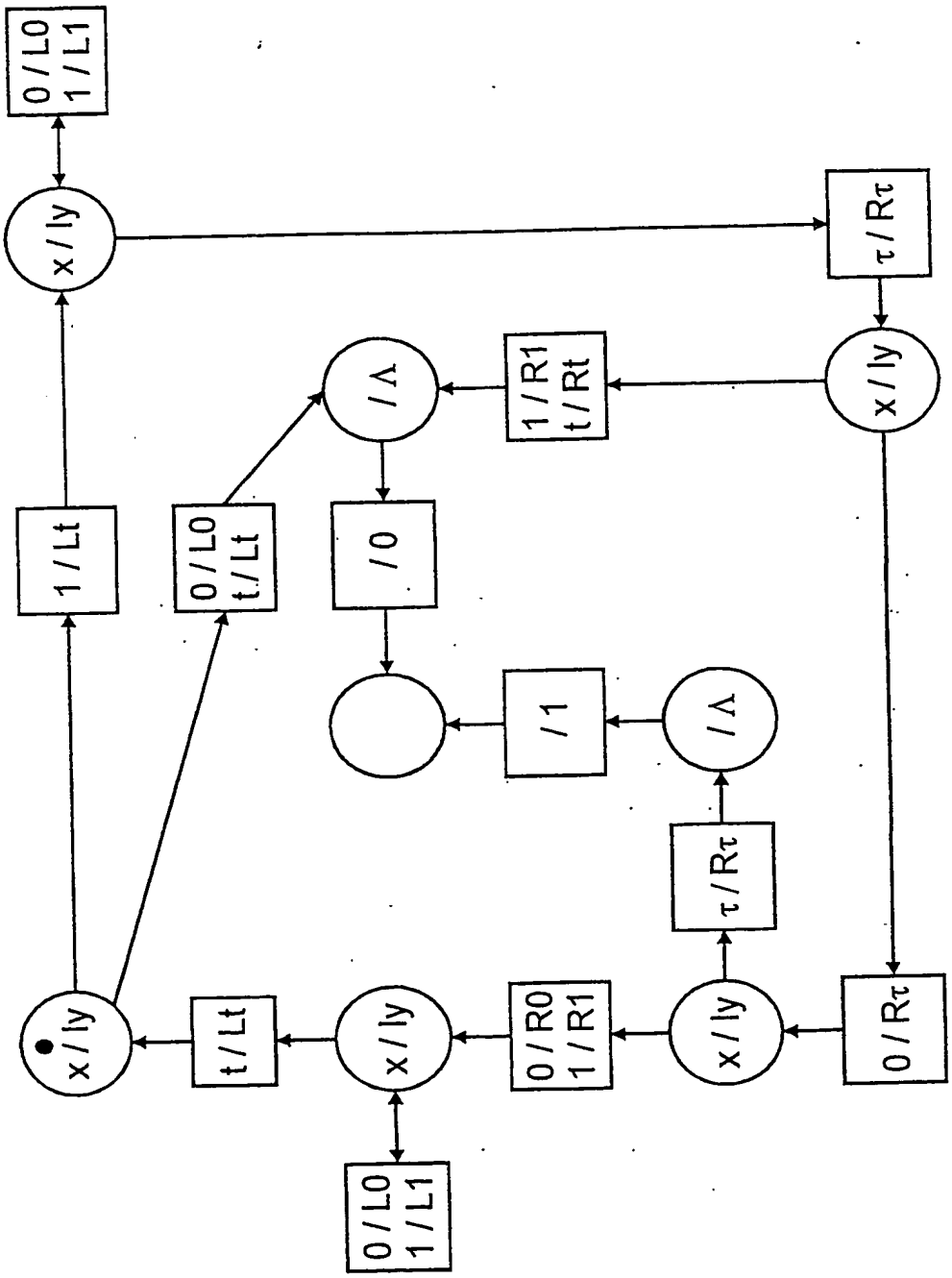

0 / L0
1 / L1
x / Iy
τ / Rτ
/ Λ
1 / R1
t / Rt
x / Iy
1 / Lt
0 / L0
t / Lt
/ 0
/ 1
/ Λ
τ / Rτ
x / Iy
t / Lt
x / Iy
0 / R0
1 / R1
x / Iy
0 / Rτ
0 / L0
1 / L1

Fig. 23

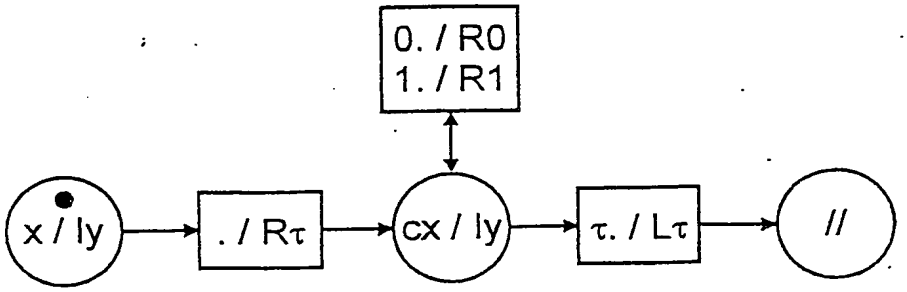


Fig. 24

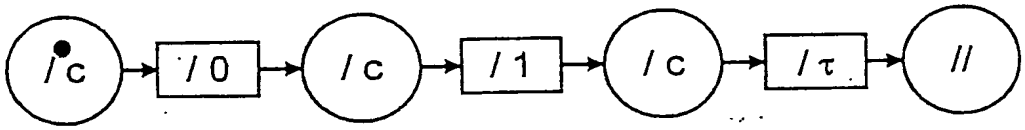


Fig. 25

Fig. 26

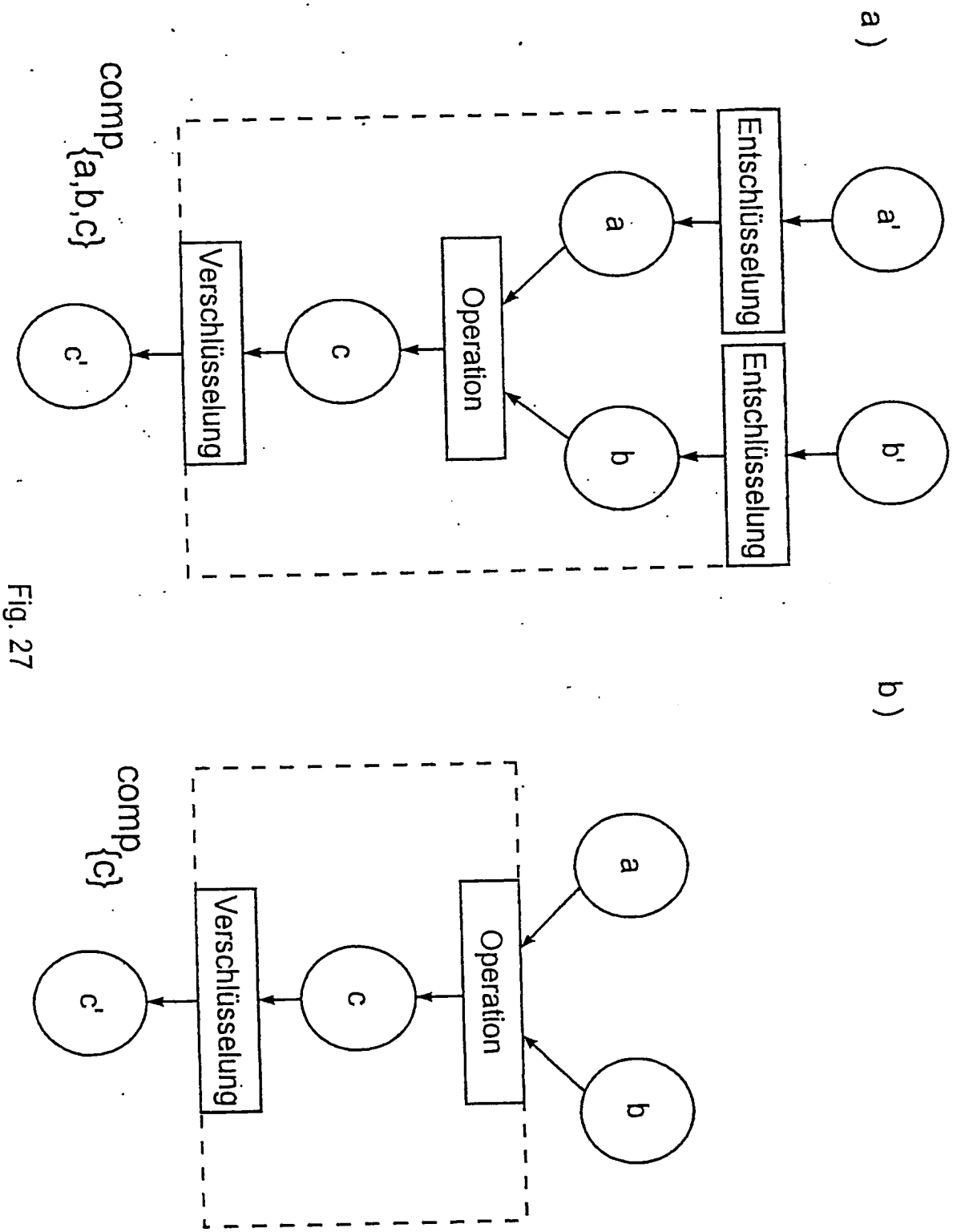
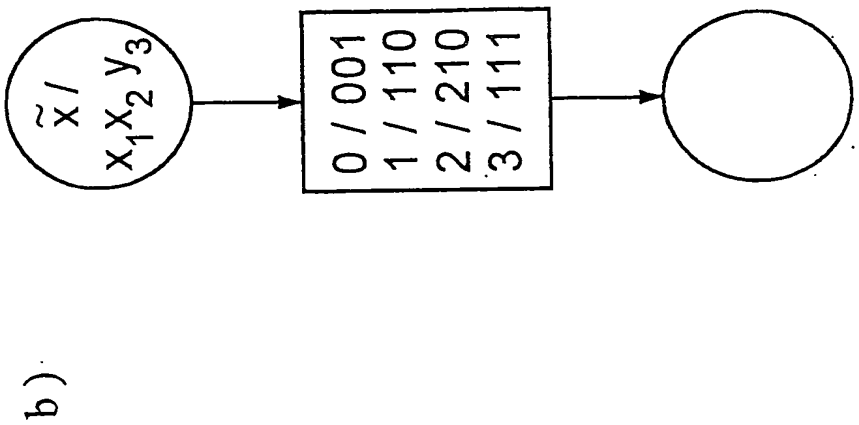
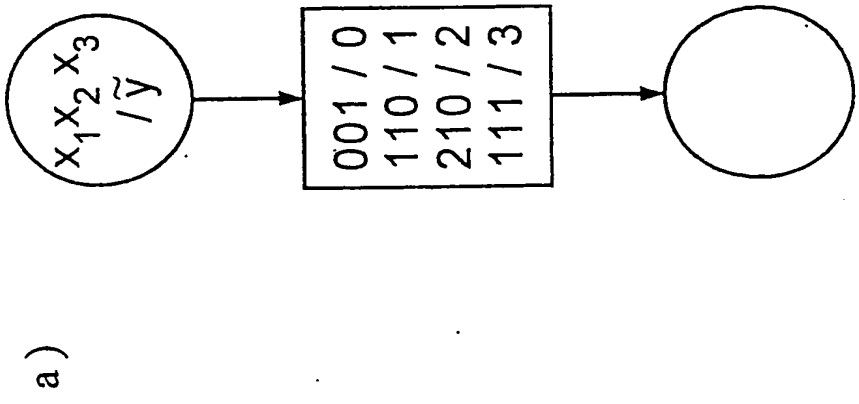

a)
comp{a,b,c}
a'
b'
Entschlüsselung
Entschlüsselung
a
b
Operation
c
Verschlüsselung
c'
b)
comp{c}
a
b
Operation
c
Verschlüsselung
c'

Fig. 27


$\tilde{x}$ /
$x_1 x_2 y_3$
0 / 001
1 / 110
2 / 210
3 / 111

b)


$x_1 x_2 x_3$
/ $\tilde{y}$
001 / 0
110 / 1
210 / 2
111 / 3

a)

Fig. 28

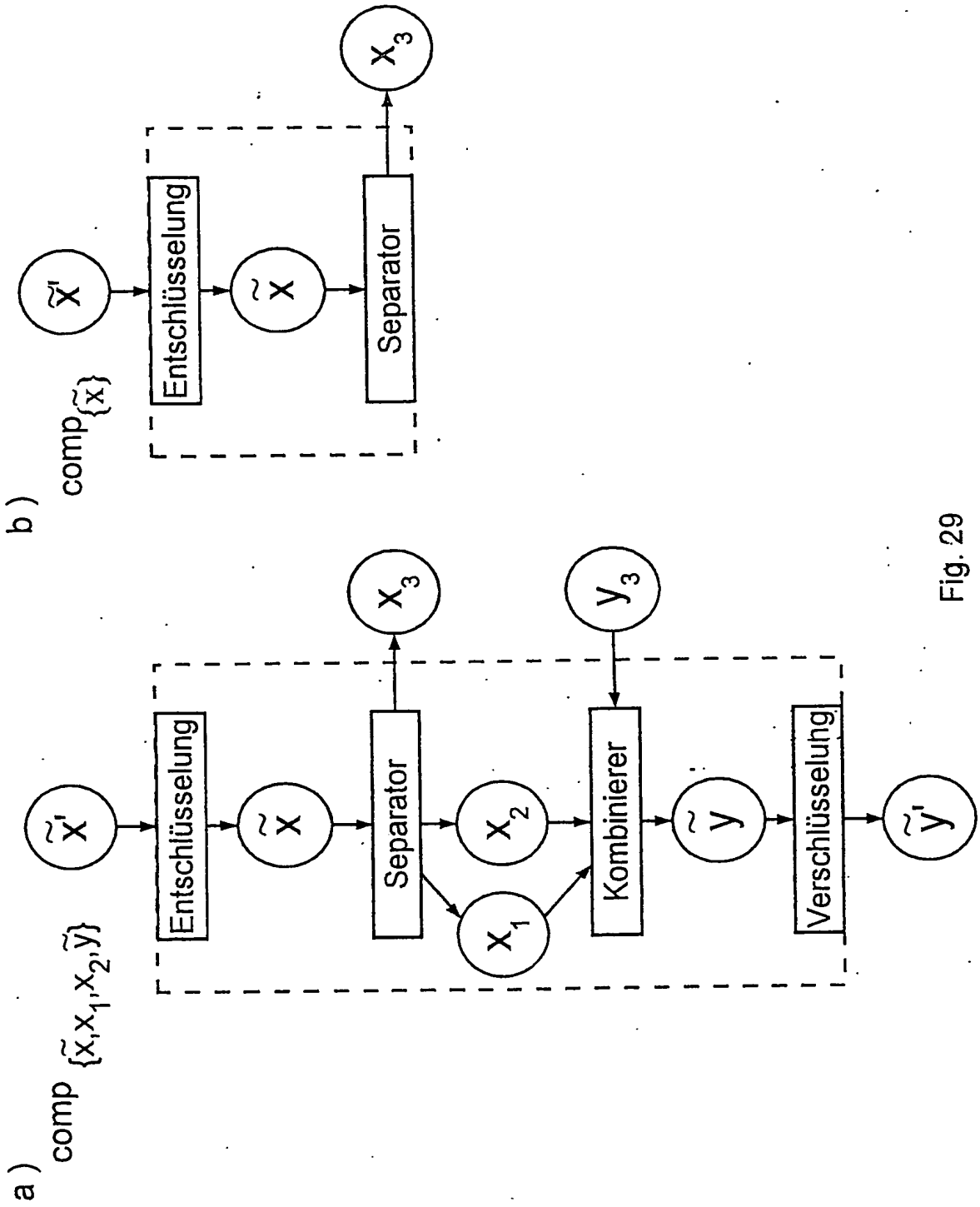

a)
comp {x̃,x1,x2,ỹ}
x̃'
Entschlüsselung
x̃
Separator
x3
x2
x1
y3
Kombinierer
ỹ
Verschlüsselung
ỹ'
b)
comp{x̃}
x̃'
Entschlüsselung
x̃
Separator
x3

Fig. 29

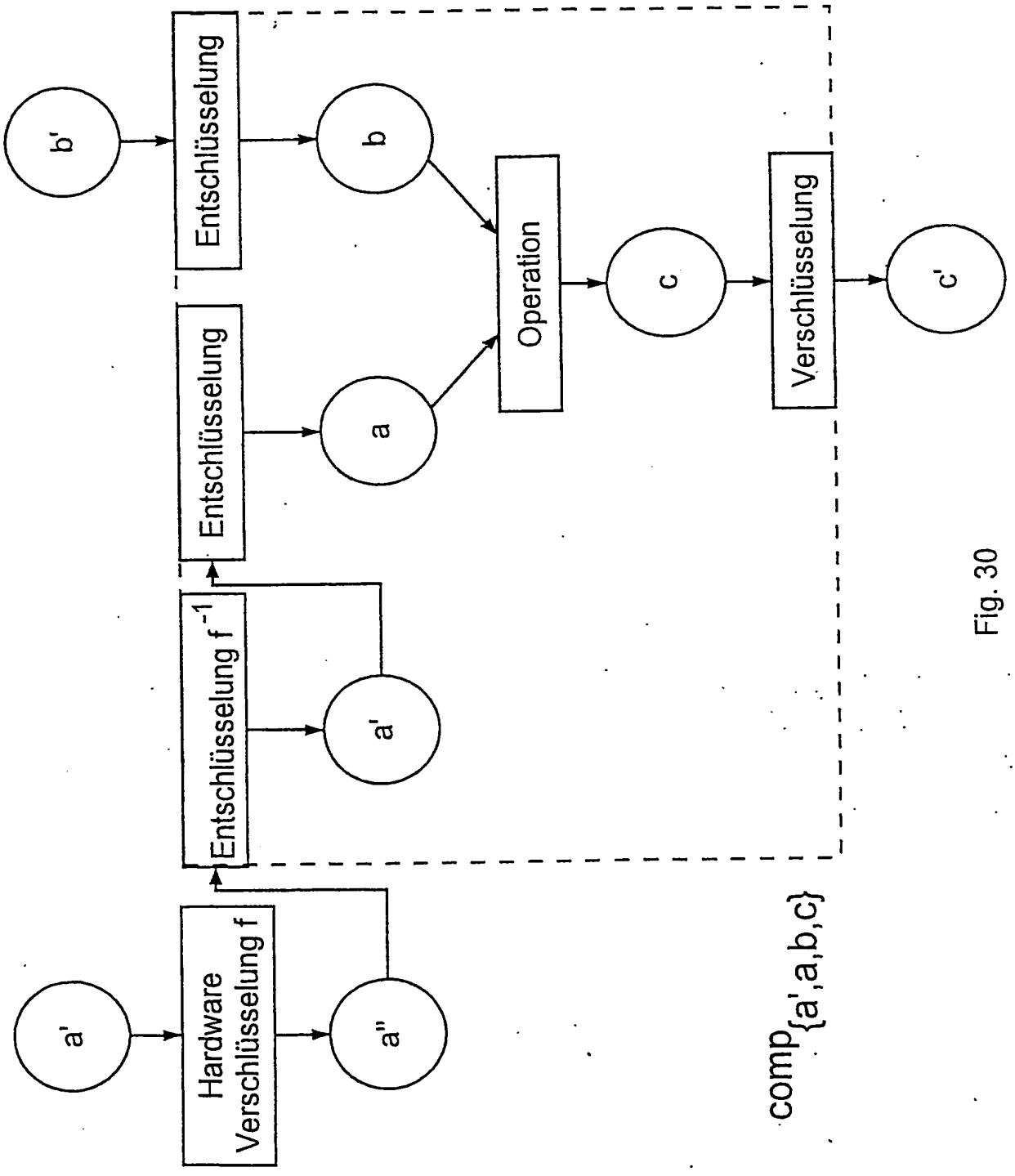

a'
Hardware Verschlüsselung f
a''
Entschlüsselung f^{-1}
a'
Entschlüsselung
a
Entschlüsselung
b'
b
Operation
c
Verschlüsselung
c'
$comp_{\{a',a,b,c\}}$

Fig. 30

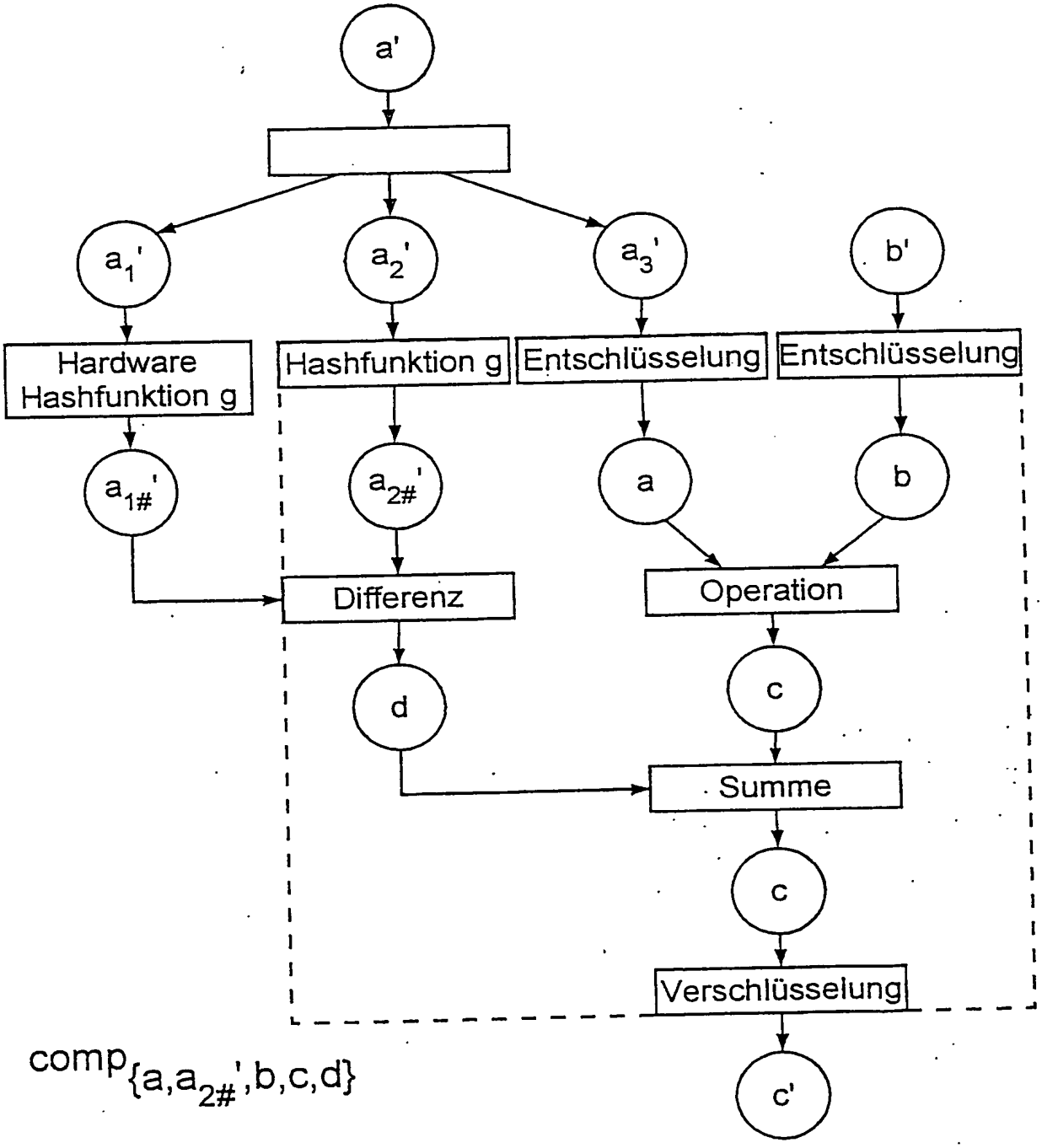

a'
a1'
a2'
a3'
b'
Hardware Hashfunktion g
Hashfunktion g
Entschlüsselung
Entschlüsselung
a1#'
a2#'
a
b
Differenz
Operation
d
c
Summe
c
Verschlüsselung
c'
comp{a,a2#',b,c,d}

Fig. 31

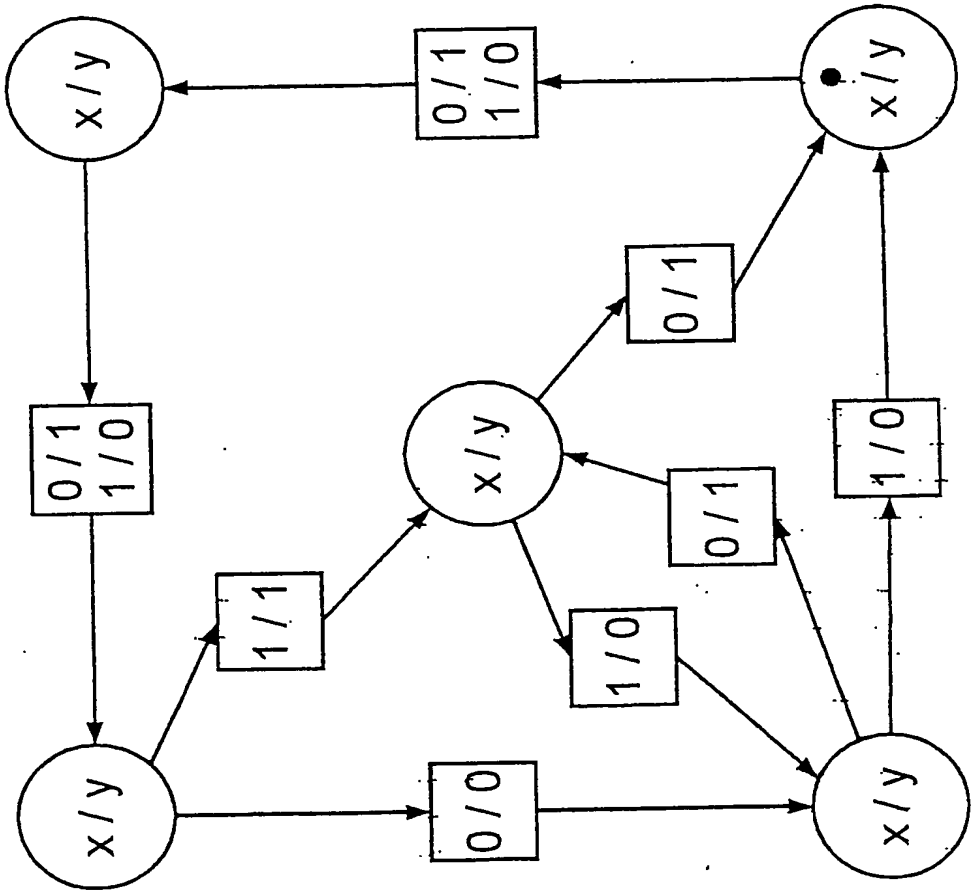


Fig. 32